Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZFI-BSC-2010-11

Bachelorarbeit

im Studiengang "Angewandte Informatik"

Referenzimplementierung einer Machbarkeitsstudie zur Visualisierung und Validierung geografischer Objekte

Felix Schindler, Andreas Strey

in der

Arbeitsgruppe Datenbanken und Informationssysteme

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

31. August 2010

Georg-August-Universität Göttingen
Zentrum für Informatik

Goldschmidtstraße 7
37077 Göttingen
Germany

Tel. +49 (5 51) 39-17 2010
Fax +49 (5 51) 39-1 44 15
Email office@informatik.uni-goettingen.de
WWW www.informatik.uni-goettingen.de

Wir erklären hiermit, dass wir die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet haben.

Göttingen, den 31. August 2010

Bachelorarbeit

Referenzimplementierung einer Machbarkeitsstudie zur Visualisierung und Validierung geografischer Objekte

Felix Schindler, Andreas Strey

31. August 2010

Betreut durch Prof. Dr. Wolfgang May
Arbeitsgruppe für Datenbanken und Informationssysteme
Georg-August-Universität Göttingen

Danksagung

Hiermit möchten wir uns besonders bei Herrn Prof. Dr. Wolfgang May für seine hervorragende Betreuung und seine zahlreichen Anregungen bedanken. Weiterhin gilt unser Dank M. Sc. Daniel Schubert dafür, dass er sich als Koreferent zur Verfügung gestellt hat, sowie Dipl.-Ing. (FH) Gunnar Krull und Dipl.-Ing. (FH) Udo Burghardt für ihre technische Unterstützung. Abschließend bedanken wir uns für die zahlreiche Rückmeldung und Unterstützung diverser Freunde und Familienmitglieder.

Inhaltsverzeichnis

1 Einleitung **6**
- 1.1 Thema . . . 6
- 1.2 Aufbau . . . 6

2 Überblick verfügbarer Techniken **8**
- 2.1 Techniken zur Visualisierung . . . 8
  - 2.1.1 OpenLayers . . . 8
  - 2.1.2 Google Earth . . . 8
  - 2.1.3 Marble . . . 9
  - 2.1.4 NASA World Wind . . . 9
  - 2.1.5 Fazit . . . 10
- 2.2 Externe Datenquellen . . . 10
  - 2.2.1 GeoNames . . . 11
  - 2.2.2 OpenStreetMaps . . . 11
  - 2.2.3 Fazit . . . 12
- 2.3 Interaktion der Visualisierung und Datenquellen . . . 12

3 Datenquelle OpenStreetMaps **14**
- 3.1 Bezugsquellen . . . 14
- 3.2 Erweiterung der Apache Commons Compress-Bibliothek . . . 15
- 3.3 .osm Daten . . . 16
  - 3.3.1 Data Primitive node . . . 17
  - 3.3.2 Data Primitive way . . . 17
  - 3.3.3 Data Primitive relation . . . 18
- 3.4 Filterung relevanter Daten . . . 19
  - 3.4.1 Identifikation der Feature Types . . . 20
  - 3.4.2 Technische Umsetzung . . . 25
- 3.5 Abfrage von Features . . . 27
- 3.6 Anmerkungen . . . 28
  - 3.6.1 Probleme . . . 28
  - 3.6.2 Verbesserung des Laufzeitverhaltens . . . 28
  - 3.6.3 Ausblick . . . 29

4 Visualisierung **30**
4.1 World Wind . . . 30
4.1.1 OpenGL und JOGL . . . 30
4.1.2 S3TC (DXTn) . . . 31
4.1.3 Technische Grundlagen . . . 32
4.2 Technische Umsetzung . . . 33
4.2.1 Verwendete Komponenten aus World Wind . . . 33
4.2.2 Verwendete Klassenstruktur . . . 36
4.2.3 MondialLayer . . . 40
4.3 Anmerkungen . . . 42
4.3.1 Probleme . . . 42
4.3.2 Ausblick . . . 43

5 Benutzerhandbuch **45**
5.1 Apache Ant . . . 45
5.2 Erzeugung der Feature Type Dateien . . . 45
5.3 World Wind . . . 46
5.3.1 Systemanforderungen . . . 46
5.3.2 System Einstellungen . . . 46
5.3.3 Ausführung . . . 47

6 Lizenzen **50**
6.1 Apache Commons Compress . . . 50
6.2 OpenStreetMap . . . 50
6.3 World Wind . . . 50

7 Fazit **51**

Quellenverzeichnis **52**

Abbildungsverzeichnis

2.1 Interaktion der Visualisierung und Datenquellen 12
2.2 Visualisierung und Validierung der Features Berlin und Brandenburg 13

3.1 SAX Handler Klassendiagramm 26

4.1 GlobeAnnotation und GeographicText 35
4.2 Österreich mit Rekursionstiefe 2 40
4.3 Übersicht World Wind 41

5.1 World Wind Graphical User Interface 47
5.2 Graphical User Interface des Mondial Layer 48

1 Einleitung

1.1 Thema

Die Arbeitsgruppe für Datenbanken und Informationssysteme verwendet die relationale Datenbank Mondial für Forschung und Lehre. In ihr werden Informationen über geografische Objekte gesammelt und ihre Beziehungen modelliert.

Im Rahmen dieser Machbarkeitsstudie sollen verschiedene Formen der Visualisierung des in Mondial zusammengetragenen Datenbestandes erprobt werden. Für die Form der Visualisierung gilt es insbesondere die Zusammenhänge zwischen den verschiedenen Arten geografischer Objekte darzustellen. So kann es von besonderem Interesse sein, die mit einem Land in Relation stehenden Provinzen abzubilden. Auch das Zusammenströmen einzelner Flüsse kann durch die Visualisierung ausdrucksvoller als durch Datenreferenzen wiedergegeben werden. Ein weiterer Aspekt ist es, Lücken im Datenbestand von Mondial leichter ausfindig zu machen. Beispielsweise können so Länder gefunden werden, für die keine Provinzen in Mondial vorhanden sind.

Ein weiterer Punkt der Machbarkeitsstudie bildet die Validierung der in Mondial enthaltenen geografischen Koordinaten. So sollen z.B. die in der Datenbank zu einer Stadt enthaltenen Koordinaten mit Daten aus einer externen Quelle verglichen werden. Diese Quelle soll zusätzlich dazu dienen, fehlende Information zu ergänzen und Objekte, für welche Mondial keine Angaben zur geografischen Lage vorsieht, zu visualisieren. Somit lässt sich die Visualisierung und Validierung der in Mondial enthaltenen Daten nur auf Grundlage externer Datenquellen realisieren.

1.2 Aufbau

Zu Beginn dieser Arbeit soll ein Überblick über die im Rahmen dieser Studie betrachteten Technologien geschaffen werden. Sowohl Techniken zur Visualisierung geografischer Daten, als auch mögliche externe Quellen, zur Beschaffung dieser, werden analysiert. Die Betrachtungen werden jeweils mit der Wahl einer Technik abgeschlossen.

Da die Datenbeschaffung die Grundlage für die Darstellung bildet, wird die Betrachtung der Datenquelle OpenStreetMaps im darauf folgendem Kapitel dargelegt. Neben den Ausführungen zu benötigten Technologien und dem strukturellen Aufbau enthält es Hinweise zur Verarbeitung der Dateien.

Darauf aufbauend folgt die Darlegung der in World Wind realisierten Visualisierung. Begonnen wird mit einem Überblick der World Wind zugrunde liegenden Technologien. Gefolgt von einer technischen Beschreibung der Implementierung, wird das Kapitel mit themenspezifischen Anmerkungen geschlossen.

In dem Benutzerhandbuch werden detaillierte Hinweise zur Installation und Verwendung der beiden Systeme gegeben.

Abgeschlossen wird diese Arbeit durch die Erwähnung der zu berücksichtigenden Lizenzen und einem Fazit der Machbarkeitsstudie.

Die Autoren teilten die Ausarbeitung in die Themenschwerpunkte "externe Datenquellen" (Felix Schindler) und "Visualisierung" (Andreas Strey) auf.

2 Überblick verfügbarer Techniken

2.1 Techniken zur Visualisierung

Im Folgenden sollen verschiedene Techniken dargestellt werden, die im Rahmen dieser Studie zur Visualiserung geografischer Daten in Frage kommen. In Betracht kommen Web- oder Desktopanwendungen, deren Stärken und Schwächen zu analysieren sind. Als Referenz für eine Webanwendung wird nur OpenLayers näher betrachtet, weil es konzeptuell ähnlich arbeitet wie beispielsweise Google Maps. Für die Desktopapplikationen liegt das Augenmerk hauptsächlich auf ihren zugrunde liegenden Systemen und den Möglichkeiten für Entwickler, eigene Implementierungen zu entwerfen.

2.1.1 OpenLayers

OpenLayers[16] ist eine JavaScript-Bibliothek, die das dynamische Anzeigen von Karten im Webbrowser, ähnlich Google Maps, ermöglicht. Sie ist frei verfügbar und wird unter einer BSD-artigen Lizenz [1] veröffentlicht. Über eine Reihe von Schnittstellen können diverse Geodaten bezogen und eingebunden werden. Neben standardisierten Diensten wie Web Feature Services und Web Map Services können auch geschlossene Formate wie Google Maps und Yahoo Maps genutzt werden. Zusätzlich stellt die OpenLayers Bibliothek einige Steuerelemente bereit. Hierzu zählen beispielsweise Elemente zum Zoomen und Navigieren der Karte.

So lassen sich mit Hilfe dieser Bibliothek sowohl auf einfachem Weg Karten in Webseiten integrieren, als auch große kartenbasierte Dienste realisieren.

2.1.2 Google Earth

Google Earth[8] ist eines der bekanntesten Programme für die 3D Visualisierung der Erde, des Mondes und des Mars. Google stellt das Produkt unter einer eigenen Lizenz [2] zur Verfügung, die eine kostenlose Verwendung für die nicht kommerzielle Nutzung erlaubt.

Google Earth gibt es in verschieden Varianten, die auf allen gängigen Betriebssystemen lauffähig sind. Die Implementierung eigener Erweiterungen[4] für Google Earth steht jedoch nur den Anwendern von Microsoft Windows Betriebssystemen zur Verfügung. Über

[1] http://svn.openlayers.org/trunk/openlayers/license.txt
[2] http://earth.google.com/intl/de-DE/license.html

die Component Object Model[6] (COM) Schnittstelle hat der Entwickler die Möglichkeit, eigene Objekte in Google Earth darzustellen. Das Component Object Model wurde von Microsoft als plattformunabhängiges, verteiltes und objektorientiertes System entwickelt. Diese Technik ermöglicht es verschiedenen Komponenten und Prozessen einfach miteinander zu kommunizieren.

2.1.3 Marble

Marble[12] ist Teil des KDE Bildungsprojektes[3], das freie Software für den Bildungs- und Erziehungssektor entwickelt und ist unter der GNU Lesser General Public License[4] verfügbar. Wie bei Google Earth handelt es sich hier auch um eine 3D-Visualisierung der Erde, des Mondes und des Mars. Um die Hardwareanforderungen gering zu halten wird bewusst auf eine 3D-Beschleunigung verzichtet. Die nötige Skalierbarkeit ist durch die Verwendung von Vektorgrafiken realisiert. Marble wird als unabhängige Applikation entwickelt, die auf Qt4 und C++ basiert. Das Projekt bietet Versionen für Linux, Mac OSX und Windows an, die auf der Projektseite [5] heruntergeladen werden können.

Für eigene Weiterentwicklungen kann ein Qt4-Widget erstellt werden, das dann noch in die eigene Applikation integriert werden muss. Über eine vordefinierte Schnittstelle kann der Entwickler eigene Objekte auf dem 3D-Globus darstellen. Über die verschiedenen Themes werden unterschiedliche Karten dargestellt. Anwender und Entwickler können so Daten auch von anderen Anbietern (z.B. OpenStreetMap) anzeigen lassen.

2.1.4 NASA World Wind

NASA World Wind[21] ist ein Java Software Devolpment Kit (SDK), das unter dem NASA Open Source Agreement steht. Bei World Wind handelt es sich um eine 3D-Visualisierung der Erde, des Mondes, des Mars, des Jupiters und der Venus. Das benötigte Kartenmaterial wird von der NASA über einen Web Service bereitgestellt und wird automatisch bei Bedarf heruntergeladen. In dem SDK befinden sich weitere Services, wie beispielsweise ein Service der Namen geografischer Objekte auf den 3D Globus projiziert.

Das SDK basiert auf dem Sun JRE ab Version 1.5. Unter Verwendung eines Java Bindings werden die 3D-Daten in OpenGL gerendert. Die 3D-Beschleunigung benötigt immer aktuelle Treiber für die vorliegende Hardware um alle Daten korrekt darstellen zu können. Als Datenformat für die Kartenausschnitte nutzt World Wind S3TC(DXT)-komprimierte Texturen, weil diese direkt von der GPU verarbeitet werden können. Diese Funktionalität muss aber von der GPU unterstützt werden. World Wind bietet weiterhin die Möglichkeit, andere Bildformate zu verarbeiten, wenn der Web Service diese Daten bereithält. Unter

[3] http://edu.kde.org/
[4] http://www.gnu.org/licenses/lgpl.html
[5] http://edu.kde.org/marble/download.php

den genannten Bedingungen ist World Wind auf allen gängigen Betriebssystemen lauffähig.

World Wind befindet sich in ständiger Weiterentwicklung, so dass täglich eine neue Version bereitgestellt wird. Die Dokumentation ist sehr lückenhaft, was eine Entwicklung erschwert. Viele Informationen, die ein Entwickler benötigt, befinden sich im Wiki [6] oder in Form von Beispielen im Download-Paket[7]. Offene Fragen können zusätzlich im zugehörigen Forum [8] gestellt werden. Hier helfen sich die User gegenseitig, aber auch die Entwickler von World Wind bringen sich aktiv ein.

2.1.5 Fazit

Google Earth kommt für eine weitere Implementierung nicht in Betracht, weil es auf Microsoft-Betriebssysteme beschränkt ist. Ein weiteres Ausschlusskriterium ist, dass es sich hier nicht um ein Open Source Projekt handelt. World Wind erhält Marble gegenüber den Vorzug, weil Java als Grundlage eine bessere Portierbarkeit verspricht. Es ist so nicht nötig den kompletten Quellcode zu verteilen, sondern es genügt den Anwendern eine ausführbare Datei bereitzustellen. So können auch unerfahrene Anwender die Applikation nutzen ohne sich mit dem Erstellen des Programms auseinander setzen zu müssen.

Sollten die Rahmenbedingungen den Einsatz einer Wabapplikation fordern, wäre die Realisierung in ähnlicher Form mit OpenLayers möglich. Wegen des Charakters einer Referenzimplementierung wird in diesem Fall die Verwendung einer universellen Programmiersprache bevorzugt.

2.2 Externe Datenquellen

Die Visualisierung und Validierung geografischer Objekte erfolgt auf Grundlage ihrer räumlichen Lage, welche durch GPS-Koordinaten wiedergegeben wird. Da in Mondial für viele Objekte eine derartige Auszeichnung fehlt oder gar nicht vorgesehen ist, bildet die Verwendung externer Datenquellen einen wichtigen Punkt dieser Studie. So sollen in den nächsten Abschnitten zunächst zwei potentielle Datenquellen kurz vorgestellt und schließlich eine Wahl getroffen werden.

Im Folgenden, wie auch in der gesamten Arbeit soll der Begriff Feature als Abstraktion eines geografischen Objekts verstanden werden. Ein Feature repräsentiert das eigentliche Objekt durch eine Menge von Eigenschaften und wird einer dem Objekt entsprechende Klasse, dem sogenannten Feature Type, zugeordnet. Beispielsweise kann ein Feature eine

[6] http://worldwindcentral.com/wiki/Main_Page
[7] http://worldwind.arc.nasa.gov/java/
[8] http://forum.worldwindcentral.com/

Stadt durch ihren Namen, ihre Population und ihre geografische Lage beschreiben, wobei es dem Feature Type Stadt zugeordnet werden würde.

2.2.1 GeoNames

GeoNames[7] ist eine geografische Datenbank, die über acht Millionen Features enthält. Das Projekt steht unter der Creative Commons Attribution 3.0 License[9] und kann somit frei genutzt werden. Neben einem Datenbankexport steht ein Web Feature Service zur Verfügung.

Bei einem Web Feature Service (WFS) handelt es sich nach der Spezifikation[15] des Open Geospatial Consortiums[14] (OGC), einer gemeinnützigen Organisation zur Definition allgemeingültiger Standards für die Verarbeitung von geografischen Daten, um einen Dienst, der den Zugriff auf geografische Daten ermöglicht. Hierzu ist eine Reihe von Operationen definiert, die zum Zweck der Interoperabilität über das HTTP-Protokoll abgewickelt werden. So lässt sich beispielsweise über die HTTP GET-Methode folgende Anfrage nach Features mit dem Namen "london" stellen:

```
http://ws.geonames.org/search?name=london
```

2.2.2 OpenStreetMaps

OpenStreetMaps[17] (OSM) ist ein Projekt mit dem Ziel frei nutzbare Weltkarten zu erzeugen. Hierzu erfassen Freiwillige mit GPS-Geräten Straßen, Flüsse, Berge und andere Objekte, die auf den Karten dargestellt werden sollen. Anschließend müssen die gesammelten Daten als eben diese Objekte ausgezeichnet werden. Hierzu werden zum Beispiel einzelne GPS-Koordinaten zu einer Straße verbunden und mit entsprechenden Attributen wie dem Straßennamen ausgezeichnet. Für diesen Schritt steht der im Rahmen des Projektes entwickelte Java OpenStreetMap Editor[10] zur Verfügung. Neben den von Freiwilligen gesammelten Daten werden auch externe Quellen einbezogen. Unter anderem wurden dem Projekt Materialien wie Satellitenbilder von Yahoo und Datenbestände von US-Bundesbehörden überlassen. Schließlich werden alle Daten an das OSM-Projekt übertragen und dort in einer Datenbank verwaltet.

Auf Grundlage dieser Datenbasis erzeugen Anwendungen, so genannte Renderer, die Kartengrafiken. Neben dem im Rahmen des OSM-Projektes entwickeltem Osmarender[11] kommt vor allem der Renderer Mapnik[11] zum Einsatz. Die erstellten Grafiken finden in Darstellungsprogrammen, wie OpenLayers, Verwendung.

[9] http://creativecommons.org/licenses/by/3.0
[10] http://josm.openstreetmap.de
[11] http://wiki.openstreetmap.org/wiki/DE:Osmarender

Aber nicht nur die erzeugten Kartengrafiken, sondern auch der gesamte Datenbestand steht zur freien Verwendung zur Verfügung. So können weitere Anwendungen wie Routenplaner oder auch dieses Projekt auf dem Datenbestand selbst realisiert werden.

2.2.3 Fazit

Ein WFS wie GeoNames bietet den für diese Studie benötigten Funktionsumfang und ermöglicht eine Integration in beliebige Systeme durch die vorliegende Standardisierung. Allerdings werden relativ wenige Feature Types angeboten, so dass die in Mondial enthaltenen Feature Types Provinz, See und Fluss nicht zur Verfügung stehen. Darüber hinaus existieren kaum weitere freie WFS.

Im Gegensatz dazu ist die Datenbasis von OSM deutlich umfangreicher. Um einen größeren Umfang an Feature Types abzudecken ist daher OSM als externe Quelle vorzuziehen. Die abweichende Zielsetzung des OSM-Projektes zu dieser Arbeit macht es allerdings nötig, die zur Visualisierung und Validierung benötigten Daten aus der gesamten OSM-Datenbasis zu filtern.

2.3 Interaktion der Visualisierung und Datenquellen

Für die Visualisierung in World Wind werden die Mondialdaten mit der externen Quelle OSM wie in der folgenden Grafik kombiniert:

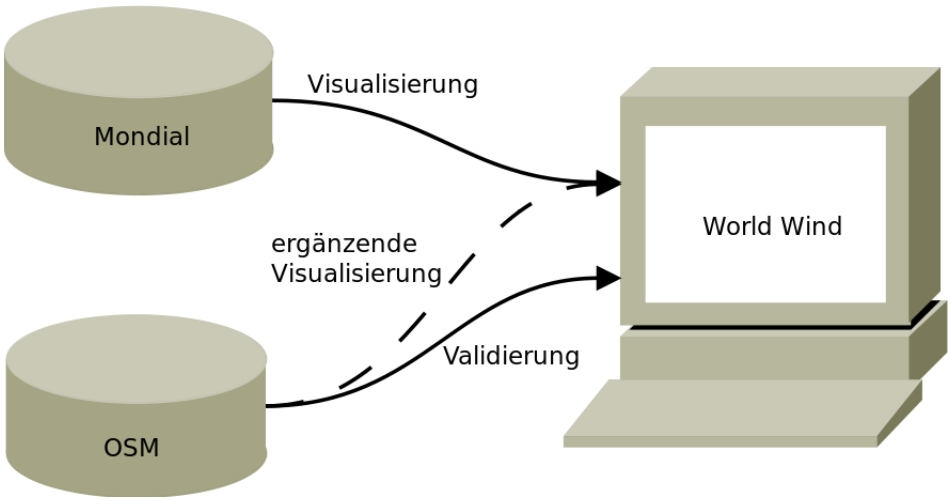

Abbildung 2.1: Interaktion der Visualisierung und Datenquellen

Die in Mondial enthaltenen Features stehen zur Visualisierung in World Wind bereit. Features, für die keine geografische Lage spezifiziert ist, werden für die Darstellung durch

Daten aus OSM ergänzt. Werden Daten aus Mondial Visualisiert, so dient diese Quelle der Validierung.

Im Folgenden wird die Visualisierung und Validierung der Features Berlin und Brandenburg dargestellt. Für den Feature Type Province sieht Mondial keine Beschreibung der geografischen Lage vor, so dass die Koordinaten der Grenzen aus OSM importiert werden. Für den Feature Type City hingegen enthält der Datenbestand von Mondial häufig die benötigten Informationen. So können diese im Fall von Berlin dargestellt und mit den Angaben aus OSM validiert werden.

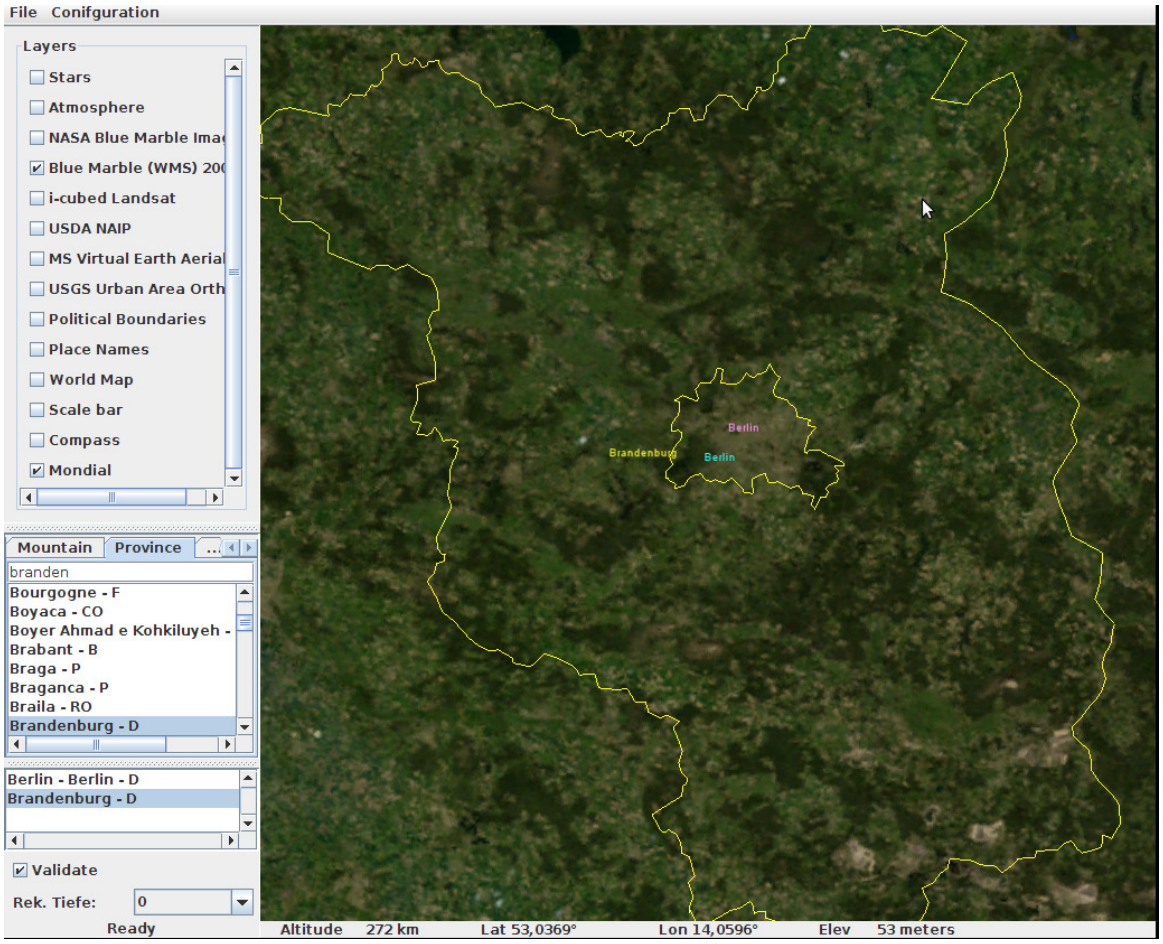

Abbildung 2.2: Visualisierung und Validierung der Features Berlin und Brandenburg

3 Datenquelle OpenStreetMaps

Neben Kartengrafiken bietet das OSM-Projekt den freien Zugriff auf die gesamte Datenbasis. In dieser können beliebige geografische Objekte enthalten sein, da die Erfassung und Auszeichnung durch die OSM-Nutzer keiner Standardisierung unterliegt. So kann es vorkommen, dass gleichartige Objekte in unterschiedlichen Auszeichnungen enthalten sind. Mangels einer ausreichenden Dokumentation ist es außerdem nicht möglich eine Aussage über die Anzahl der erfassten Feature Types zu machen.

Auf dieser Grundlage stellt die Identifikation und Klassifizierung, der im Rahmen dieser Studie relevanten Feature Types, eine große Herausforderung dar. Es ist gelungen die folgenden Feature Types herauszufiltern: Country, Province, City, River, Lake, Island und Mountain.

In diesem Kapitel werden zunächst grundlegende Themen, wie der Bezug und die Dekomprimierung, zur Verarbeitung der OSM-Daten erwähnt. Danach folgt die Filterung der für diese Studie relevanten Daten und die auf diesen Daten realisierte Abfrage von Features. Abgeschlossen wird das Kapitel mit der Erörterung einiger themenspezifischer Anmerkungen.

3.1 Bezugsquellen

Das OSM-Projekt bietet zwei Wege, auf den Datenbestand zuzugreifen. Zum einen bietet die API[1] eine Methode zum Formulieren von Abfragen, zum anderen stehen Downloadserver[2] bereit.

Über die Abfragemethode der API lassen sich rechteckige Kartenausschnitte, definiert durch einem minimalen und maximalen Breiten- und Längengrad, beziehen. Allerdings existieren starke Einschränkungen für diese Abfragen. Um die Last auf den Server zu reduzieren ist die maximale Größe eines Kartenausschnittes auf ein viertel Grad, sowohl Breiten- als auch Längengrad, beschränkt. Außerdem ist die intensive Nutzung durch häufige Abfragen untersagt.

Um größere Kartenausschnitte zu beziehen stehen neben einem eigenen Downloadserver des OSM Projektes weitere Spiegelserver zur Verfügung. Neben einem kompletten Abbild der Datenbank werden Ausschnitte, z.B. von einzelnen Ländern, angeboten.

[1] http://wiki.openstreetmap.org/wiki/API_v0.6
[2] http://wiki.openstreetmap.org/wiki/Planet.osm

Die Daten werden in beiden Fällen XML-formatiert in `.osm` Dateien ausgeliefert. Große Kartenausschnitte können schnell mehrere Gigabyte, bis hin zu 130 GB annehmen und werden daher vor der Auslieferung komprimiert.

3.2 Erweiterung der Apache Commons Compress-Bibliothek

Der Datenbestand des OSM-Projektes wird in der Regel vor der Auslieferung mit Hilfe von bzip2 komprimiert. Da für das Entpacken viel Speicherplatz nötig wäre, ist es von Vorteil direkt auf den komprimierten Daten zu arbeiten. Allerdings ist hierbei zu beachten, dass das OSM-Projekt zum Komprimieren eine parallel arbeitende Implementierung von bzip2 einsetzt. Dies hat zur Folge, dass viele Bibliotheken Schwierigkeiten im Umgang mit diesen Dateien haben, so auch die Commons Compress-Bibliothek[1] der Apache Software Foundation[3].

Bei den parallel arbeitenden Versionen von bzip2 werden die Daten in mehreren einzelnen bzip2 Streams komprimiert und diese aneinandergereiht. Die Commons Compress-Bibliothek hingegen erwartet einen einzigen Stream, der alle Daten enthält. So schließt sie die Verarbeitung mit dem Ende des ersten Streams ab, ohne die Daten der darauf folgenden Streams zu berücksichtigen.

Die Erweiterung der Commons Compress-Bibliothek ist im `BZip2InputStream`[4] implementiert. Sie erweitert hierzu die Abstrakte Klasse `CompressorInputStream`[5] und bietet somit einen `InputStream`[6] zum Lesen aus bzip2-komprimierten Daten.

Genau wie bei der Klasse `BZip2CompressorInputStream`[7] erwartet der Konstruktor einen `InputStream` zum Lesen der komprimierten Daten. Doch im Gegensatz zu dieser Klasse beendet die Erweiterung ihre Arbeit bei parallel komprimierten Daten nicht bereits nach dem ersten bzip2-Stream. Es wird nacheinander für jeden in der Datei enthaltenen bzip2-Stream ein `BZip2CompressorInputStream` erzeugt, der nach wie vor die eigentliche Dekomprimierung übernimmt.

Somit ermöglicht die Erweiterung der Apache Commons Compress-Bibliothek den Streambasierten Zugriff auf Dateien, die mit einer parallel arbeitenden Version von bzip2 komprimiert wurden. Die Filterung der relevanten Daten kann somit über eine Streamverarbeitung realisiert werden, ohne eine vorherige Dekomprimierung vorauszusetzen.

[3] http://www.apache.org
[4] de.uni_goettingen.informatik.dbis.compressors.BZip2InputStream
[5] org.apache.commons.compress.compressors.CompressorInputStream
[6] java.io.InputStream
[7] org.apache.commons.compress.compressors.bzip2.BZip2CompressorInputStream

3.3 .osm Daten

Für die Verarbeitung der `.osm` Daten gilt es zunächst die verwendete Struktur zu analysieren, welche im Folgenden beschrieben wird.

Die geografische Beschaffenheit von Features wird im OSM-Datenbestand mit Hilfe von drei Objektarten modelliert, einem Punkt-Objekt, einem Linien-Objekt und einem Objekt zur Beschreibung von Beziehungen.

Einzelne Orte, wie Briefkästen oder Telefonzellen, werden durch ihre GPS Koordinaten repräsentiert und in Form eines Punkt-Objektes modelliert. Die Punkt-Objekte dienen darüber hinaus dazu lineare Objekte aufzubauen. Beispielsweise wird ein Feature wie ein Fluss durch mehrere, den Flussverlauf wiederspiegelnde Punkt-Objekte innerhalb eines Linien-Objektes modelliert. Zusätzlich können zusammengesetzte und abstrakte Features in Form von Beziehungen modelliert werden. Hierzu fasst ein weiteres Objekt die einzelnen Objekte, aus denen das Feature besteht, zusammen. So kann eine Buslinie modelliert werden, indem alle von dem Bus befahrenen Straßen zusammengefasst werden.

Diese drei Objektarten werden in den `.osm` Dateien in der Extensible Markup Language[8] durch die Strukturelemente `node`, `way` und `relation` beschrieben und im Folgenden als Data Primitives bezeichnet. Die Referenzen innerhalb der Data Primitives lassen sich über die Strukturelemente `nd` und `member` realisieren. Das `ref`-Attribut der `nd`-Elemente dient der Referenzierung von `nodes` innerhalb von `way`-Primitives. Innerhalb von `relation`-Primitives findet das `member`-Element Verwendung. Das `ref`-Attribut gibt die Referenz und das `type`-Attribut den referenzierten Typ an.

Außerdem können die drei Data Primitives mit Eigenschaften ausgezeichnet werden. Dies geschieht durch ein weiteres Strukturelement, den `tag`. Jeder `tag` besteht aus einem Schlüssel-Wert Paar.

```
<tag k="key" v="value" />
```

Weder für den Schlüssel noch für den Wert existieren generelle Einschränkungen. Dennoch gibt es eine Reihe von Empfehlungen[9] zur Auszeichnung einzelner Feature Types durch bestimmte `tag` Elemente. Dies soll eine einheitliche Interpretation und Darstellung der Data Primitives ermöglichen.

Nach der Auflistung der gemeinsamen Attribute folgt eine nähere Betrachtung jedes einzelnen Data Primitives.

id gibt eine eindeutige Identifikationsnummer innerhalb gleichartiger Data Primitive an.

user, uid geben den Namen und die Identifikationsnummer des Benutzers an, welcher zuletzt das entsprechende Element geändert hat.

[8] http://www.w3.org/XML

[9] http://wiki.openstreetmap.org/wiki/Map_Features

timestamp gibt den Zeitpunkt der letzten Bearbeitung an.

visible gibt an, ob das Element zum aktuellen Datenbestand gehört oder nur zur Abfrage alter Datenbestände weiterhin existiert.

version ist ein inkrementeller Zähler der Änderungen des Elementes.

changeset gibt die Identifikationsnummer des Arbeitsschrittes an, in welchen das Element geändert wurde.

3.3.1 Data Primitive node

Ein `node`-Data Primitiv spiegelt einen geografischen Punkt wieder und bildet die wesentliche Grundlage für den Aufbau der weiteren Data Primitives. Neben den oben genannten besitzt es noch die Attribute `lat` und `lon` zur Angabe des Breiten- und Längengrades. `Node`-Elemente werden zum einen dazu verwendet, das Data Primitive `way` aufzubauen, können aber auch alleinstehende Punkte wie Verkehrszeichen, Telefonzellen und Briefkästen kennzeichnen. Diese Punkte werden durch entsprechende `tag`-Elemente ausgezeichnet:

```
<node id="360755773"
      lat="51.5663944"
      lon="9.9268568"
      user="Alexander␣Eickhoff"
      uid="24646"
      visible="true"
      version="1"
      changeset="814811"
      timestamp="2009-03-15T16:22:16Z">
   <tag k="amenity" v="post_box"/>
</node>
```

In einigen Fällen ist auch die Kombination der beiden Verwendungsarten möglich. So kann ein `node`-Primitive zum Beispiel sowohl dazu verwendet werden, die Auszeichnung einer Buslinie aufzubauen als auch alleinstehend eine Bushaltestelle zu kennzeichnen.

3.3.2 Data Primitive way

Das Data Primitiv `way` dient der Darstellung von linearen Objekten wie Straßen, Flüssen und Eisenbahnstrecken. Zur geografischen Auszeichnung linearer Objekte fassen `way`-Primitives eine geordnete Menge von `node`-Elementen zusammen. Die `node`-Elemente spiegeln den eigentlichen Verlauf des Objektes wieder und werden hierbei über das Struktur-

element nd referenziert. Ergänzend definieren tag-Elemente Eigenschaften wie die Art und den Namen des Objektes.

```
<way id="35825958"
     user="Alexander␣Eickhoff"
     uid="24646"
     visible="true"
     version="1"
     changeset="1489786"
     timestamp="2009-06-11T19:00:28Z">
  <nd ref="319192091"/>
  <nd ref="294043358"/>
  <nd ref="294043357"/>
  <nd ref="64763477"/>
  <nd ref="64763482"/>
  <nd ref="64763483"/>
  <nd ref="64763502"/>
  <nd ref="94699070"/>
  <nd ref="64763510"/>
  <nd ref="64763515"/>
  <tag k="highway" v="residential"/>
  <tag k="name" v="Goldschmidtstrasse"/>
</way>
```

Stimmen das erste und letzte referenzierte node-Element überein, so handelt es sich um einen geschlossenen Weg. Dieser dient der Darstellung von Flächen wie Parkplätzen, Seen und Parks.

3.3.3 Data Primitive relation

In dem Data Primitive relation lassen sich andere Primitives zusammenfassen um Beziehungen und abstrakte Objekte zu modellieren. Häufig beschreiben way-Primitives nur Teilabschnitte von z.B. Straßen und Flüssen, da ein Limit von 2000 nd-Referenzen pro way-Element existiert. Außerdem ermöglicht eine höhere Granularität eine detaillierte Beschreibung durch tag-Elemente, wie das Auszeichnen eines Einbahnstraßen-Abschnittes einer Straße. Diese Teilabschnitte können im Data Primitive relation zusammengefasst und dadurch die gesamte Straße aus Teilabschnitten modelliert werden.

Aber auch abstrakte Dinge wie Buslinien lassen sich darstellen. Hierbei wird die Busstrecke aus way-Primitives der entsprechenden Straßenabschnitte modelliert. Einzelne node-Primitives zeichnen die Haltestellen aus.

```
<relation id="54072"
```

```
        user="mmertens"
        uid="18369"
        visible="true"
        version="90"
        changeset="4751114"
        timestamp="2010-05-19T19:48:12Z">
    <member type="node" ref="316813487" role=""/>
    ...
    <member type="node" ref="316922160" role=""/>
    <member type="way" ref="8093142" role=""/>
    ...
    <member type="way" ref="38074073" role=""/>
    <tag k="network" v="VSN"/>
    <tag k="note" v="Nikolausberg - Ellershausen"/>
    <tag k="operator" v="GoeVB"/>
    <tag k="ref" v="5"/>
    <tag k="route" v="bus"/>
    <tag k="type" v="route"/>
</relation>
```

Wie die anderen Data Primitives enthält auch das `relation`-Primitive beliebige `tag`-Elemente. Die Referenzen auf andere Data Primitive werden über das `member`-Element realisiert. Die `type`- und `ref`-Attribute stellen Verweise auf Primitives des angegebenen Typus mit der referenzierten `id` her.

Das Attribut `role` ermöglicht zusätzlich noch die Rolle des referenzierten Primitives zu spezifizieren. In dem Beispiel der Buslinie könnte mit den Attributen `role="forward"` und `role="backward"` gekennzeichnet werden, ob die Fahrtrichtung des Busses der Richtung des referenzierten `way`-Elementes entspricht oder entgegengesetzt verläuft.

Auch bei durch `relation` ausgezeichneten Flächen kommt das `role` Attribut häufig zum Einsatz. So z.B. bei der Auszeichnung von Verwaltungsgebieten. Hierbei werden die Außengrenzen durch das Attribut `role="outer"` gekennzeichnet. Die Abgrenzung von Enklaven hingegen geschieht über das Attribut `role="inner"`.

3.4 Filterung relevanter Daten

Die Datenbasis von OSM ist im Vergleich zu den im Rahmen dieses Projektes verwendeten Feature Types deutlich umfangreicher. Um die Datenmenge auf eine von gängigen Desktopsystemen handhabbare Menge zu reduzieren und damit die Abfrage von Features für ein interaktives System nutzbar zu machen, gilt es nur die benötigten Data Primitives herauszufiltern. In den folgenden Abschnitten werden zunächst die im Rahmen dieser

Studie entwickelten Kriterien zur Identifikation der einzelnen Feature Types beschrieben. Darauf aufbauend wird die technische Umsetzung der Filterung mit Hilfe dieser Kriterien dargelegt.

3.4.1 Identifikation der Feature Types

Alle Data Primitives werden über das `tag`-Strukturelement beschrieben und so ihre Eigenschaften ausgewiesen. Bei der Identifikation von bestimmten Feature Types gilt es folglich die Primitives anhand ihrer `tag`-Elemente zu filtern. Da keine Einschränkungen bei der Wahl des Schlüssels und Wertes existieren, ist es schwer, generelle Regeln zur Identifikation von einzelnen Feature Types anzugeben. Anhand der Empfehlungen[10] zur Vergabe von `tag`-Elementen und mit Hilfe von Beispielen ist es dennoch gelungen, für die Praxis funktionstüchtige Filter zu erstellen. Es folgt eine Auflistung der Kriterien zur Identifikation der einzelnen Feature Types.

Feature Type Country

Die geografische Repräsentation des Feature Types Country erfolgt durch die Außengrenzen des Landes. Grenzen eines Verwaltungsgebietes werden durch das `tag`[11]-Element

```
<tag k="boundary" v="administrative"/>
```

ausgezeichnet. Die Ebene der Verwaltungseinheit wird durch ein weiteres `tag`-Element gekennzeichnet. Es wird entweder der Schlüssel `k="admin_level"` oder `k="border_type"` genutzt. Für den Schlüssel `k="admin_level"` nimmt der Wert eine Ganzzahl zwischen eins und zehn an. Der Wert `v="2"` kennzeichnet die nationalen Grenzen. Bei dem Schlüssel `k="border_type"` kann der Wert `v="nation"` oder `v="country"` nationale Grenzen kennzeichnen.

Der Feature Type Country wird in der Regel durch das Primitive `relation` modelliert, welches weitere `way`-Primitives referenziert. In diesem Fall stellen die `way`-Elemente die Grenzabschnitte zu den einzelnen Nachbarstaaten dar. Zusätzlich wird in einigen Fällen noch die Hauptstadt des Landes in Form eines `node`-Elementes referenziert.

```
<relation id="50046"
          user="GeorgFausB"
          uid="72151"
          visible="true"
          version="62"
          changeset="5452880"
```

[10] http://wiki.openstreetmap.org/wiki/Map_Features
[11] http://wiki.openstreetmap.org/wiki/Key:admin_level#admin_level

```
        timestamp="2010-08-10T11:08:54Z">
    <member type="node" ref="13707878" role="capital"/>
    <member type="way" ref="48921087" role=""/>
    ...
    <member type="way" ref="48925554" role=""/>
    <tag k="admin_level" v="2"/>
    <tag k="boundary" v="administrative"/>
    <tag k="ISO3166-1" v="dk"/>
    <tag k="name" v="Danmark"/>
    <tag k="name:en" v="Denmark"/>
    ...
    <tag k="name:es" v="Dinamarca"/>
    <tag k="type" v="boundary"/>
</relation>
```

Feature Type Province

Die Auszeichnung von Provinzen ähnelt stark dem Feature Type Country. Sie unterscheidet sich lediglich in den Werten für den Schlüssel `k="admin_level"` bzw. `k="border_type"`. Der Schlüssel `k="admin_level"` hat den Wert `v="4"`. Der Schlüssel `k="border_type"` den Wert `v="region"` oder `v="state"`.

```
<relation id="454192"
          user="mikes"
          uid="6270"
          visible="true"
          version="60"
          changeset="5431931"
          timestamp="2010-08-08T11:21:34Z">
    <member type="way" ref="29413021" role="outer"/>
    ...
    <member type="way" ref="29413247" role="outer"/>
    <tag k="admin_level" v="4"/>
    <tag k="boundary" v="administrative"/>
    <tag k="de:amtlicher_gemeindeschluessel" v="03"/>
    <tag k="name" v="Niedersachsen"/>
    <tag k="type" v="multipolygon"/>
</relation>
```

Feature Type City

Der Feature Type City wird durch ein `node`-Primitiv modelliert, welches das Stadtzentrum repräsentiert. Es kann durch das `tag`-Element

```
<tag k="place" v="city"/>
```

identifiziert werden.

```
<node id="240114841"
      lat="51.5336849"
      lon="9.9345070"
      user="ToB"
      uid="15766"
      visible="true"
      version="14"
      changeset="3205803"
      timestamp="2009-11-24T16:54:08Z">
   <tag k="is_in" v="Niedersachsen,Deutschland,Europe"/>
   <tag k="name" v="Goettingen"/>
   <tag k="name:de" v="Goettingen"/>
   ...
   <tag k="name:pl" v="Getynga"/>
   <tag k="place" v="city"/>
   <tag k="population" v="122187"/>
</node>
```

Die Existenz eines `tag`-Elementes mit dem Schlüssel `k="is_in"` ist bei diesem Feature Type optional. Es dient lediglich der genaueren Beschreibung der entsprechenden Stadt und kann nicht als Referenz auf weitere Data Primitives verstanden werden. Der Wert dieses `tag`-Elementes wird dennoch bei der Suche von Features berücksichtigt.

Feature Type River

Beim Feature Type River wird der Flussverlauf entweder direkt durch das `way`-Primitive oder durch mehrere `way`-Primitives in einem `relation`-Element modelliert. Die Data Primitives lassen sich anhand des Wertes `v="river"` identifizieren, welcher aber in Kombination mit verschiedenen Schlüsseln auftreten kann. In der Regel wird der Schlüssel `k="waterway"` genutzt. Insbesondere bei den `relation`-Primitives finden aber auch die Schlüssel `k="collection"` und `k="route"` Verwendung.

```
<relation id="123688"
          user="aighes"
```

```
            uid="110639"
            visible="true"
            version="22"
            changeset="4513322"
            timestamp="2010-04-24T16:28:52Z">
    <member type="way" ref="51528588" role=""/>
    ...
    <member type="way" ref="4439473" role=""/>
    <tag k="destination" v="Aller"/>
    <tag k="name" v="Leine"/>
    <tag k="type" v="waterway"/>
    <tag k="waterway" v="river"/>
</relation>
```

Feature Type Lake

Der Feature Type Lake wird entweder durch ein `way`- oder ein `node`-Primitive modelliert. Bei Seen, die detaillierter erfasst sind, repräsentiert das `way`-Primitive die Uferlinie. In einigen Fällen werden sogar `relation`-Primitives verwendet. Der Feature Type lässt sich anhand des `tag`-Elementes

```
<tag k="natural" v="water"/>
```

identifizieren. Auf diese Weise werden allerdings alle Arten von Wasserflächen ausgezeichnet. So fallen sowohl kleinste künstliche Teiche und Auffangbecken als auch große Seen darunter. Zur Klassifizierung gibt es keine direkte Möglichkeit. Um nicht zu viele kleine Wasserflächen zu filtern, empfiehlt es sich die Existenz eines `tag`-Elementes mit dem Schlüssel `k="name"` vorauszusetzen. Die namentliche Erfassung garantiert in der Regel, dass es sich um eine hinreichend große und bedeutende Wasserfläche wie beispielsweise einen See handelt.

```
<way id="54685299"
     user="KK-O"
     uid="14228"
     visible="true"
     version="6"
     changeset="5431065"
     timestamp="2010-08-08T09:46:40Z">
   <nd ref="694712100"/>
   ...
   <nd ref="694712100"/>
```

```
    <tag k="boat" v="yes"/>
    <tag k="is_in" v="Deutschland, Oesterreich, Schweiz"/>
    <tag k="name" v="Bodensee"/>
    <tag k="name:cs" v="Bodamske jezero"/>
    ...
    <tag k="name:pt" v="Lago de Constanca"/>
    <tag k="natural" v="water"/>
    <tag k="wikipedia:de" v="Bodensee"/>
</way>
```

Feature Type Island

Der Feature Type Island wird durch ein `node`-Primitive modelliert, welches einen Punkt auf der Insel repräsentiert. Es kann durch das `tag`-Element

```
<tag k="place" v="island"/>
```

identifiziert werden.

```
<node id="208865934"
      lat="54.9256495"
      lon="8.3408698"
      user="Spongilla"
      uid="25729"
      visible="true"
      version="3"
      changeset="5072188"
      timestamp="2010-06-25T09:33:10Z">
   <tag k="name" v="Sylt"/>
   <tag k="name:dk" v="Sild"/>
   <tag k="place" v="island"/>
</node>
```

Feature Type Mountain

Der Feature Type Mountain wird durch ein `node`-Primitive modelliert, welches den Berggipfel repräsentiert. Es kann durch das `tag`-Element

```
<tag k="natural" v="peak"/>
```

identifiziert werden. Über ein `tag`-Element mit dem Schlüssel `k="ele"` wird häufig noch die Höhe des Berges angegeben.

```
<node id="26862634"
      lat="51.7991241"
      lon="10.6155733"
      user="Ebbe73"
      uid="55521"
      visible="true"
      version="16"
      changeset="4964280"
      timestamp="2010-06-11T18:02:06Z">
   <tag k="ele" v="1141"/>
   <tag k="name" v="Brocken"/>
   <tag k="natural" v="peak"/>
</node>
```

3.4.2 Technische Umsetzung

Die im Rahmen dieser Arbeit entwickelte Anwendung zur Filterung relevanter Daten baut auf den zuvor entworfenen Kriterien zur Identifikation der Feature Types auf. Diese ermöglicht alle für die Visualisierung und Validierung benötigten Daten aus dem Bestand des OSM-Projektes zu filtern.

Aufgrund der Größe ist es nicht möglich, den gesamten Datenbestand im Speicher zu halten um beispielsweise eine DOM-orientierte[12] API zu verwenden. Daher muss auf eine sequenzielle Verarbeitung wie die Simple API for XML (SAX) zurückgegriffen werden.

Ein SAX-Parser arbeitet ereignisorientiert. Es ist eine Menge von Ereignissen definiert, die bei der sequenziellen Verarbeitung der Daten, beim Auftreten einer entsprechenden Struktur, ausgelöst werden. Die wichtigsten mit SAX anfallenden Ereignisse (`startDocument`, `startElement`, `characters` ...) sind in dem Interface `org.xml.sax.ContentHandler` definiert, welche vom DefaultHandler[13] implementiert wird.

Als abstrakte Oberklasse der Implementierung dient der OSMHandler[14], welcher vom DefaultHandler erbt. Die Klasse stellt grundlegende Funktionalitäten zum Parsen der Data Primitives in `.osm`-Dateien bereit. Zusätzlich enthält sie Methoden zur Identifikation der Feature Types gemäß den oben genannten Kriterien.

Vom OSMHandler werden drei weitere Klassen abgeleitet, der RelationHandler[15], der WayHandler[16] und der NodeHandler[17]. Jeder der drei Handler ist für die Verarbeitung

[12] http://www.w3.org/DOM
[13] org.xml.sax.helpers.DefaultHandler
[14] de.uni_goettingen.informatik.dbis.osm.OSMHandler
[15] de.uni_goettingen.informatik.dbis.osm.RelationHandler
[16] de.uni_goettingen.informatik.dbis.osm.WayHandler
[17] de.uni_goettingen.informatik.dbis.osm.NodeHandler

des entsprechenden Data Primitives zuständig.

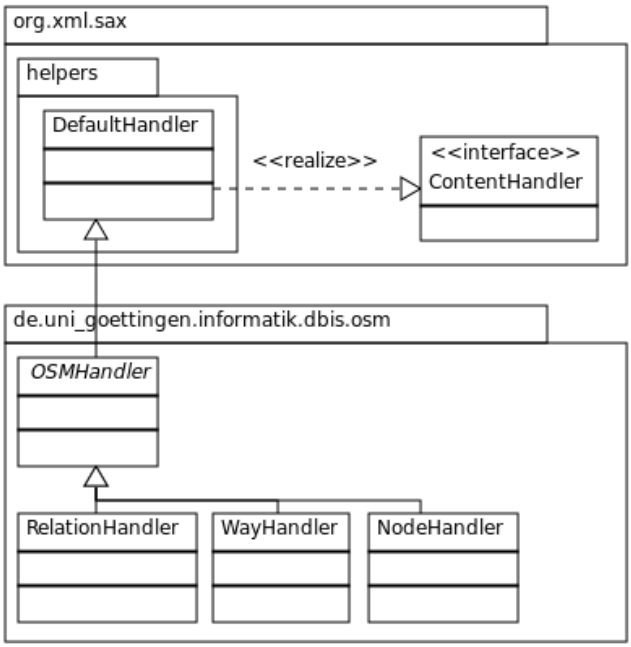

Abbildung 3.1: SAX Handler Klassendiagramm

Neben dem Standardkonstruktor erben die drei Handler zusätzlich einen weiteren Konstrukter des OSMHandlers, welcher die Übergabe von `way`- und `node`-IDs ermöglicht. So können die Handler über Primitives informiert werden, die von bereits identifizierten Features referenziert werden. Diese sollen auch dann gefiltert werden, wenn sie nicht direkt als einer der Feature Types identifiziert werden.

In der Main-Klasse[18] wird der benötigte Stream zum Lesen der komprimierten `.osm` Datei, der SAX Parser und die drei Handler initialisiert. Um die Referenzen der Data Primitives untereinander auflösen zu können muss der Parser die Daten dreimal durchlaufen. Zunächst kommt der `RelationHandler`, dann der `WayHandler` und schließlich der `NodeHandler` zum Einsatz. Jeder Handler parst die entsprechenden Primitives, prüft sie auf relevante Feature Types und schreibt sie gegebenenfalls in die Ausgabedateien.

Wird ein `relation`-Primitive als relevantes Feature Type identifiziert, so werden alle enthaltenen Referenzen auf weitere Data Primitives gespeichert. Bei `way`-Primitives hingegen wird nur jede dreißigste `nd`-Referenz gespeichert. Dadurch wird der Detaillierungsgrad der linearen Objekte reduziert. So wird die Größe der Ausgabedateien minimiert und ein

[18] de.uni_goettingen.informatik.dbis.osm.Main

besseres Laufzeitverhalten erzielt. Die gespeicherten Referenzen werden dem darauf folgendem Handler, mit Hilfe des parametrisierten Konstruktors übergeben. Dies stellt sicher, dass alle referenzierten Primitives in den Ausgabedateien enthalten sind.

Die gefilterten Daten werden auf mehrere Ausgabedateien aufgeteilt, so dass für jeden Feature Type eine eigene Datei entsteht. Um Laufzeitvorteile bei der Abfrage von Features zu erzielen, kommt ein leicht abgewandeltes `.osm` XML-Schema[19] zum Einsatz. Es werden alle Bezeichner von Strukturelementen und Attributen auf eine minimale Länge reduziert. Zusätzlich wird ein Großteil der Attribute, welche für die weitere Verarbeitung irrelevant sind, gelöscht.

3.5 Abfrage von Features

Die Visualisierung und Validierung geografischer Objekte setzt die Suche nach den entsprechenden Features voraus. Daher ist in der Klasse `GetFeature`[20] die Suche nach Features implementiert. Unter Angabe des entsprechenden Feature Types kann nach dem Namen eines Features gesucht werden. Präzisierend kann der Name eines Verwaltungsgebietes, innerhalb dessen das Feature liegt, angegeben werden.

Für jede Suchanfrage wird die entsprechende Feature Type-Datei durchlaufen und alle enthaltenen Data Primitives untersucht. Hierbei fällt das Augenmerk auf die drei `tag`-Elemente mit den Schlüsseln `k="name"`, `k="name:en"` und `k="name_int"`, in welchen sprachabhängige Varianten des Namens enthalten sind. Auf Grundlage dieser wird für jedes Data Primitive die Relevanz in Bezug auf den Suchbegriff berechnet.

Wurde ein Verwaltungsgebiet des Features angegeben und ist ein `tag`-Element mit dem Schlüssel `k="is_in"` vorhanden, so kann die Relevanz des entsprechenden Data Primitives dementsprechend erhöht werden.

Die Berechnung der Relevanz beruht auf der Genauigkeit der Übereinstimmung von Suchbegriff und Wert der `tag`-Elemente. Hierfür werden drei Fälle unterschieden, wobei in allen Fällen die Groß- und Kleinschreibung irrelevant ist. Dem ersten Fall werden nur exakte Übereinstimmungen zugeordnet. Der zweite Fall deckt Übereinstimmungen unter Berücksichtigung von verschiedenen Trennzeichen und Umlauten ab. Die beiden Trennzeichen Bindestrich und Leerzeichen werden als gleichbedeutend betrachtet. Zusätzlich können Umlaute durch ihre entsprechenden Vokale substituiert werden. Der dritte Fall stellt die ungenaueste Überprüfung dar. Unter Verwendung der Regeln des zweiten Falles wird hierbei sogar nur auf das Enthaltensein des Begriffes überprüft. Um bei extrem kurzen Begriffen nicht zu ungenaue Ergebnisse zu liefern werden nur Begriffe, die aus mindestens vier Zeichen bestehen, berücksichtigt.

[19] siehe type.dtd

[20] de.uni_goettingen.informatik.dbis.osm.GetFeature

Das Ergebnis der Suchanfrage setzt sich aus den Data Primitives mit der höchsten Relevanz zusammen, wobei bei `relation`- und `way`-Primitives die Referenzen entsprechend aufgelöst werden.

Die Suche bietet zusätzlich die Möglichkeit nach mehreren Features desselben Feature Types gleichzeitig zu suchen. So muss die entsprechende Feature Type-Datei nicht für jedes gesuchte Feature erneut durchlaufen werden, sondern es reicht ein einziger Durchlauf aus.

3.6 Anmerkungen

3.6.1 Probleme

Bei der Verarbeitung von Daten des OSM-Projektes stellt besonders die fehlende Standardisierung eine große Hürde dar. Da die Schlüssel-Wert Paare beim Auszeichnen von Data Primitives keiner Restriktion unterliegen und frei gewählt werden können, ist es oft schwer die einzelnen Feature Types zu identifizieren. Häufig sind viele Probeläufe und Anpassungen nötig um einen bestimmten Feature Type herauszufiltern.

Der große Datenumfang von über 100 GB führt zu zwei weiteren Nachteilen. Zum einen beansprucht das Parsen derartiger Datenmengen extrem lange Laufzeiten, zum anderen wird der Einsatz von sequenziellen Parsern nötig, die nicht die komplette Datenstruktur im Speicher halten und somit mehrere Durchgänge zum Auflösen von Referenzen benötigen.

Da sich das OSM-Projekt noch in der Entwicklung befindet und maßgeblich von der Aktivität der Benutzer abhängt, weist der Datenbestand teilweise größere Lücken auf. Vor allem in Gebieten mit geringer Bevölkerungsdichte und niedrigerem Technologisierungsgrad sind weniger Daten erfasst. So kann die Dichte und Genauigkeit der aus den `.osm` Daten gewonnen Features regionalen Schwankungen unterliegen. Durch erneutes Erstellen der Feature Type-Dateien aus aktuellen OSM-Datenbeständen können neu erfasste Features gefiltert und somit verfügbar gemacht werden.

Einige von Mondial geforderte Feature Types erweisen sich als schwierig zu filtern. Dies ist darauf zurückzuführen, dass das primäre Ziel des OSM-Projektes die Erstellung von Karten ist. Somit sind vor allem geografisch großflächige und schwach abgegrenzte Feature Types wie Kontinente, Meere und Wüsten schwer zu identifizieren oder nicht vorhanden.

3.6.2 Verbesserung des Laufzeitverhaltens

Da die Suche nach Features auf die interaktive Verwendung ausgelegt ist, spielt das Laufzeitverhalten eine besondere Rolle. Einen Großteil der Laufzeit wird bei der Suche in den Dateien eingebüßt. Um die Geschwindigkeitseinbußen zu reduzieren, werden bei der Filterung die relevanten Daten auf mehrere Dateien aufgeteilt, so dass für jeden Feature Ty-

pe eine separate Datei entsteht. Dadurch müssen in Abhängigkeit der gesuchten Feature Types nur deutlich kleinere Dateien durchlaufen werden. Um darüber hinaus auch die Häufigkeit, mit der die Feature Type-Dateien durchlaufen werden müssen, zu reduzieren, ist die Suche so implementiert, dass nach mehreren Features in einem Durchlauf gesucht werden kann.

Um die Größe der Feature Type-Dateien weiter zu reduzieren und somit die Suche zu beschleunigen, wird zum einen eine deutlich kompaktere Version des `.osm` XML-Schemas verwendet und zum anderen der Detaillierungsgrad von linearen Objekten reduziert.

3.6.3 Ausblick

Durch den Einsatz einer Datenbank könnten sich einige Vorteile erzielen lassen. Das häufige Parsen der Feature Type-Dateien würde entfallen und es könnten Indizes erstellt werden. So ließe sich die Abfrage von Features mit einem deutlich besseren Laufzeitverhalten realisieren.

Es würde sich anbieten den Einsatz einer derartigen Datenbank zu zentralisieren, um die Benutzer von der Last zu befreien die Feature Type-Dateien zu erstellen und vorzuhalten. Dies bietet zusätzlich den Vorteil, dass der Dienst leicht von anderen Anwendungen genutzt werden könnte.

Für diesen Zweck müsste die Implementierung lediglich um einen einfachen Web Service erweitert werden. Dieser könnte den Datenzugriff in Form eines Web Feature Services ermöglichen um die Interoperabilität zu gewährleisten.

4 Visualisierung

Die Visualisierung wird in dieser Studie mit Hilfe von World Wind realisiert. Zunächst folgt ein kurzer Einblick in die Geschichte und der zugrunde liegenden Technologien von World Wind. Anschließend wird die technische Umsetzung der Implementierung vorgestellt. Ergänzt wird das Kapitel mit einer Übersicht der aufgetretenen Probleme und einem kurzen Ausblick für weiterführende Entwicklungen.

4.1 World Wind

World Wind wurde im Herbst 2004 unter der NASA Open Source Agreement v1.3 veröffentlicht. In diesem Stadium der Entwicklung handelte es sich noch um eine eigenständige Anwendung, die unter dem .NET Framework und DirectX lief. Aufgrund dieser Systemanforderungen war World Wind nur unter Microsoft ab Windows 2000 lauffähig. Die Applikation bot in dieser Version bereits Funktionen für die Darstellung von KML- [1] und Shapefiles [2] und das Einbinden von Web Map Services und Web Feature Services. Für die Entwicklung von Add-ons wurde eine Schnittstelle auf Basis des .NET Framework bereitgestellt.

Am 8. Mai 2007 wurde das Java SDK veröffentlicht. World Wind ist mit dieser Umstellung keine eigene Applikation mehr, sondern dazu gedacht die Funktionen von World Wind in eigene Applikationen zu integrieren. Durch die Umstellung auf Java wurde eine einfache Portierbarkeit zwischen verschieden Systemen geschaffen, die gute Möglichkeiten für Multiplatformapplikationen bietet. Ein weiterer Schritt war die Änderung der Programmierschnittstelle DirectX zu OpenGL. Im Gegensatz zu DirectX ist OpenGL eine plattform- und programmiersprachenunabhängige Programmierschnittstelle, die eine Portierung überhaupt erst ermöglicht.

4.1.1 OpenGL und JOGL

OpenGL[19] ist eine Spezifikation für eine plattform- und programmiersprachenunabhängige Programmierschnittstelle zur Entwicklung von 2D- und 3D-Computergrafiken. In der

[1] http://earth.google.de/userguide/v4/ug_kml.html
[2] www.esri.com/library/whitepapers/pdfs/shapefile.pdf

Regel werden die Funktionen in Systembibliotheken oder Grafikkarten-Treibern implementiert. Unterstützte Befehle werden direkt auf der Grafikhardware ausgeführt um teure Berechnungen auf der CPU zu vermeiden.

OpenGL wurde als Zustandsautomat entworfen, der benötigte Parameter so lange weiter verwendet, bis diese sich verändern. Der große Vorteil dieser Implementierung ist, dass teure Reorganisationen der Grafikpipeline so lange wie möglich hinausgezögert werden. Für die Entwicklung mit OpenGL erspart es dem Entwickler aufwendige Angaben von Parametern, die sich für viele Objekte nicht ändern. Das bedeutet für die Wartung und Wiederverwendung des Programmcodes einen Mehrwert für die Entwickler.

Bei JOGL[9] handelt es sich um eine Wrapperbibliothek, die in der Regel nativen C-Code im Hintergrund ausführt. Die Bibliothek wurde in Zusammenarbeit von Sun Microsystems und SGI im Jahr 2003 ins Leben gerufen. Ziel war es, der Spieleindustrie die Möglichkeit zu geben, auch die Programmiersprache Java für die Spieleentwicklung benutzen zu können. Es wurde versucht, die Vorteile aus bereits bestehenden OpenGL-Bindings in JOGL zu verbinden, um eine stabile und performante Bibliothek zu erzeugen.

Für die Zukunft ist es geplant, dass JOGL ein fester Bestandteil des Sun JDK werden soll. So wird die Entwicklung von 3D beschleunigten Anwendungen in Java weiter vereinfacht. Die aktuelle Weiterentwicklung wird von der Game Technology Group von Sun Microsystems voran getrieben.

4.1.2 S3TC (DXTn)

S3TC[20] ist ein Texturkomprimierungssystem von S3 Graphics. Im Vergleich zu anderen Bildkompressionsalgorithmen wie JPEG eignet es sich für beschleunigte Computergrafiken, weil es eine feste Datenkompressionsrate hat und pro Texel nur einen Zugriff benötigt. S3TC wurde fester Bestandteil von DirectX in Version 6.0. Bis zum heutigen Tag hat sich S3TC zum vorherrschenden Standard für die Speicherung von komprimierten Texturen entwickelt. Für kommende OpenGL Versionen ist es geplant, dass S3TC fester Bestandteil der Spezifikation werden soll. Bis dahin muss die Verarbeitung über Erweiterungen [3] realisiert werden.

Die Texturen können mit Hilfe von fünf verschiedenen Kompressionsalgorithmen erzeugt werden, wobei der Hauptunterschied in der Handhabung des Alphakanals liegt. DXT1 z.B. benutzt nur ein Bit für den Alphakanal, wodurch die Sichtbarkeit des Pixels deaktiviert oder aktiviert wird. DXT3 besitzt einen expliziten Alphakanal und DXT5 versucht zwischen undurchsichtigen und durchsichtigen Pixeln zu interpolieren um den Übergang möglichst weich darstellen zu können. In der heutigen Zeit finden DXT2 und DXT4 kaum noch Verwendung. Die mögliche Kompressionsrate liegt bei DXT1 bei 8:1 und bei den anderen vier Verfahren bei 4:1.

[3] http://www.opengl.org/registry/specs/EXT/texture_compression_s3tc.txt

World Wind nutzt die Texturen um die Kartenausschnitte auf dem 3D-Globus zu platzieren. Beliebige Web Dienste stellen die Texturen im Direct Draw Surface-Format (.dds) [4] zum Download zur Verfügung. Anschließend wird der 3D-Globus mit den heruntergeladenen Texturen überzogen. Diese Vorgehensweise hat den Vorteil, dass aufwendige Rechenoperationen für die Darstellung von Karten entfallen.

Weiterhin bekommt World Wind durch die Verwendung von S3TC anstelle von normalen Grafiken einen betrachtlichen Geschwindigkeitsschub. Der Grund dafür ist, dass die Texturen direkt auf der GPU verarbeitet werden können und teure Rechenoperationen auf der CPU vermieden werden. Der Flaschenhalseffekt bei der Übertragung der Daten zur Grafikhardware wurde ebenfalls reduziert, da die Texturen eine geringere Größe aufweisen als normale Grafiken.

4.1.3 Technische Grundlagen

Die erste minimale Anwendung mit World Wind besteht aus einem `WorldWindowGLCanvas` [5], dem ein `BasicModel` [6] (siehe Kapitel 4.2.1) zugewiesen wird. Der `Canvas` ist anschließend auf einem `Frame` zu platzieren. Das Ergebnis ist eine Anwendung, die einen 3D-Globus der Erde anzeigt. Dabei wird die Standardkonfiguration aus dem `worldwind.jar` verwendet. Für weitere Anpassungen kann ein eigener Ordner `config` im gleichen Verzeichnis angelegt werden, in dem die eigenen Konfigurationsanpassungen möglich sind.

Layer

Layer repräsentieren in World Wind die Implementierung einer oder mehrerer Funktionalitäten. So existieren bereits Layer, die für die Darstellung von Kartentexturen zuständig sind. Andere Layer sorgen für die Abbildung bestimmter geografischer Objekte wie beispielsweise Ländergrenzen (`Political Boundaries`) oder Namen der Objekte (`Place Names`).

Die zur Laufzeit zur Verfügung stehenden Layer werden von World Wind als seperate Darstellungsschichten betrachtet. Jede dieser Schichten kann je nach Bedarf aktiviert oder deaktiviert werden. Die Abbildung der einzelnen Schichten wird mit jeder Mausbewegung neu erzeugt, was kurze Ausführungszeiten für die Implementierung eines Layers fordert um fließende Bewegungen darstellen zu können.

[4] http://msdn.microsoft.com/en-us/library/bb943990%28VS.85%29.aspx
[5] gov.nasa.worldwind.awt.WorldWindowGLCanvas
[6] gov.nasa.worldwind.BasicModel

Konfiguration

Über die Datei `worldwind.xml` werden die Grundeinstellungen für World Wind vorgenommen. Hier werden Optionen wie z.B. die initiale Entfernung des Blickpunktes vom Globus und die Verwendung von bestimmten Klassen, die für das Rendern der Daten benötigt werden, voreingestellt. Ein Beispiel dafür ist die `Property <name="gov.nasa.worldwind.avkey.GlobeClassName">`. Diese Option hat Auswirkung auf das eigentlich dargestellte geometrische Objekt. Es existieren Klassen, die die Karten als Fläche oder Kugel darstellen. Zusätzlich entscheidet diese Option über den zu verwendenden Planeten.

Über die `<LayerList>` können die zu verwendenden Layer angeben werden. So wird es ermöglicht alle Services zu laden, die die Anwendung benötigt. Je nach Struktur des Layer wird hier direkt auf die Klasse des Layer oder auf eine separate Konfigurationsdatei verwiesen. In der beiliegenden Datei `worldwind.layers.xml` findet man die Standardlayer für eine einfache Implementierung einer World Wind Anwendung.

Im Folgendem eine Liste mit den möglichen Elementen einer `<LayerList>`:

classname direkte Angabe der Klasse mit dem kompletten Paket, z.B. `gov.nasa.worldwind.layers.StarsLayer`

href Pfad zu einer separaten Konfigurationsdatei des Layers

title der Titel des Layers, wobei je nach Implementierung der Name über die Methode `toString()` der Klasse geladen wird

actuate legt den Startstatus des Layers fest. `onLoad` aktiviert den Layer direkt beim Start. `onRequest` nimmt den Layer in die verfügbaren Layer auf, muss allerdings über die Methode `setEnabled(boolean enable)` aktiviert werden.

Weitere Einstellungen können während der Laufzeit über die Klasse `Configuration`[7] vorgenommen werden. Hierbei handelt es sich um eine statische Klasse, die bei der ersten Verwendung automatisch die Konfigurationsdateien liest. Die Klasse bietet Methoden zum Lesen und Ändern der Parameter. Genauere Informationen sind im Javadoc der Klasse zu finden.

4.2 Technische Umsetzung

4.2.1 Verwendete Komponenten aus World Wind

Für diese Studie wird eine Reihe von Komponenten aus World Wind benötigt. Die Auflistung zusammen mit ihren Beschreibungen soll aufzeigen, welche Möglichkeiten für die Entwicklung zur Verfügung stehen und wie sie eingesetzt werden können.

[7] gov.nasa.worldwind.Configuration

WorldWindowGLCanvas [8] ist eine heavyweight AWT Komponente, die für die Darstellung des `Model`[9] mit dem beinhalteten `Globe` und der `Layer` zuständig ist. Es bildet das Grundgerüst für jede World Wind-Applikation und basiert auf dem `GLCanvas`[10] um die 3D-Daten darzustellen.

Globe [11] repräsentiert den darzustellenden Globus. Implementierungen dieses `Interface` stellen die verschiedenen abzubildenden geometrischen Formen und Planeten dar. Ein `Globe` kann dazu als Ellipsoid oder Fläche dargestellt sein. Auf dem definierten geometrischen Objekt können anschließend die Texturen platziert werden. Es existieren bereits verschieden Klassen für die Erde, Mars und Mond.

BasicModel [12] ist eine Implementierung des Interfaces `Model`. Um die Anwendungsdaten unabhängig von der eigentlichen Darstellung halten zu können, wird diese Klasse verwendet. In ihr werden die `Layer` und der `Globe` gespeichert und verwaltet.

Layerpanel [13] ist ein mitgeliefertes Beispiel, das vom `JPanel` [14] erbt. Über die zugehörigen Checkboxen ist es möglich die jeweiligen Layer während der Programmausführung zu aktivieren oder deaktivieren

Statusbar [15] ist ebenfalls ein mitgeliefertes Beispiel. Sie dient der Anzeige aktueller Statuinformationen, zu denen die Entfernung des Blickpunktes zum Globus, die aktuelle Position des Mauszeigers in Längen- und Breitengrad und den aktuellen Downloadstatus einzelner Services gehört.

Layer müssen von dem `AbstractLayer` [16] erben. Es wird die Methode

```
public void doRender(DrawContext dc)
{
        // TODO Auto-generated method stub
}
```

implementiert, in der die darzustellenden Objekte des Layers für das Rendern vorbereitet werden.

Position [17] repräsentiert einen Punkt auf dem Globus. Sie bietet Möglichkeiten um bei-

[8] gov.nasa.worldwind.awt.WorldWindowGLCanvas
[9] gov.nasa.worldwind.Model
[10] javax.media.opengl.GLCanvas
[11] gov.nasa.worldwind.globes.Globe
[12] gov.nasa.worldwind.BasicModel
[13] gov.nasa.worldwind.examples.LayerPanel
[14] javax.swing.JPanel
[15] gov.nasa.worldwind.util.Statusbar
[16] gov.nasa.worldwind.layers.AbstractLayer
[17] gov.nasa.worldwind.geom.Position

spielsweise die Entfernung zwischen 2 Punkten zu berechnen.

GeographicText [18] repräsentiert eine Textauszeichnung, die an einer angegebenen `Position` auf dem Globus dargestellt wird.

GeographicTextRenderer [19] nimmt Objekte vom Typ `GeographicText` auf und rendert diese.

GlobeAnnotation [20] repräsentiert ein Textlabel, das an der angegebenen Position auf dem Globus platziert wird. Sie unterscheidet sich vom `GeographicText` in der Form, dass es nicht nur ein reiner Text auf dem Globus ist sondern eine Auszeichnung in Form einer Sprechblase darstellt, die in dieser Applikation beim Überfahren eines `GeographicText` mit der Maus angezeigt wird. In der folgenden Abbildung wird zur besseren Unterscheidung eine `GlobeAnnotation` und ein `GeographicText` dargestellt.

Abbildung 4.1: GlobeAnnotation und GeographicText

Die Formatierung des dargestellten Textes erfolgt über einen geringen Satz an HTML-Elementen. In der verwendeten World Wind Version befinden sich keine Informationen über die implementierten Elemente, allerdings ist es bekannt, dass die Elemente `<p>`, `<br>`, `<b>`, `<i>` korrekt interpretiert werden.

BasicAnnotationRenderer [21] nimmt Objekte vom Typ `Annotation`[22] auf und rendert diese.

Polyline [23] ist ein Linienzug, der aus mindestens zwei Punkten besteht. Eigenschaften wie Farbe, Linienbreite usw. lassen sich individuell anpassen. Es werden weitere Funktionen bereitgehalten, die z.B. die Länge der Linie ermitteln oder die Linie verschieben.

[18] gov.nasa.worldwind.render.GeographicText
[19] gov.nasa.worldwind.render.GeographicTextRenderer
[20] gov.nasa.worldwind.render.GlobeAnnotation
[21] gov.nasa.worldwind.render.BasicAnnotationRenderer
[22] gov.nasa.worldwind.render.Annotation
[23] gov.nasa.worldwind.render.Polyline

Angle [24] repräsentiert den geometrischen Winkel. Einfache mathematische Funktionen wie z.B. Sinus, Kosinus und Addition sind in der Klasse ebenfalls vorhanden.

Sector [25] repräsentiert eine Boundingbox, die durch Angabe der minimalen und maximalen Breiten- und Längengrade aufgespannt wird. Für die Erzeugung eines `Sectors` wird erwartet, dass die Breitengrade auf den Bereich +/- 90 und Längengrade auf den Bereich +/- 180 normalisiert sind. Mit Hilfe dieser Box kann die Sichtbarkeitsprüfung für `Position` und `Polyline` durchgeführt werden.

PositionEvent [26] beinhaltet die aktuelle und vorhergehende Position des Mauszeigers umgerechnet auf Breiten- und Längengrad. Zusätzlich liefert das Event die Position des Mauszeigers in absoluten Koordinaten innerhalb des registrierten Objektes mit dem Koordinatenursprung in der oberen linken Ecke.

PositionListener [27] gehört zur Gruppe der EventListener. Alle Klassen, die diesen Listener implementieren, müssen die Methode

```
public void moved(PositionEvent event)
{
        // TODO Auto-generated method stub
}
```

umsetzen. Um die nötigen Events der Mausbewegung zu empfangen muss der Listener zusätzlich am `WorldWindowGLCanvas` registriert werden.

DrawContext [28] bildet die Kernkomponente für die Darstellung der Objekte. Er übernimmt entweder direkt oder indirekt (z.B. über den `GeographicTextRenderer` [29]) das Rendern der benötigten Objekte.

4.2.2 Verwendete Klassenstruktur

Die zugrunde liegende Klassenstruktur wurde auf Grundlage des ER-Diagramms [30] von Mondial entworfen. Sie dient als interne Datenhaltung für die vorhandenen Features. In dem Paket `de.uni_goettingen.informatik.dbis.worldwind.objects` befinden sich die Klassen für die einzelnen Feature Types aus Mondial, die alle von dem Grundtyp `GeoObject` erben.

[24] gov.nasa.worldwind.geom.Angle
[25] gov.nasa.worldwind.geom.Sector
[26] gov.nasa.worldwind.event.PositionEvent
[27] gov.nasa.worldwind.event.PositionListener
[28] gov.nasa.worldwind.render.DrawContext
[29] gov.nasa.worldwind.render.GeographicTextRenderer
[30] http://www.dbis.informatik.uni-goettingen.de/Teaching/DBP-files/mondial-ER.pdf

GeoObject

Das `GeoObject` definiert für alle erbenden Objekte die größtmögliche Schnittmenge, die aus einer ID, einem Namen und Koordinaten besteht. Um die Darstellung in World Wind zu realisieren kommen folgende Objekte aus dem Java SDK zum Einsatz:

Der `Sector` dient der Sichtbarkeitsprüfung der darzustellenden Objekte und wird durch die aufgespannte Boundingbox des `GeoObject` definiert. Er muss dazu die Schnittmenge mit dem `visibleSector` des `DrawContext` berechnen.

Ein `GeoObject` kann durch einen einzigen Punkt dargestellt werden. Zu diesem Punkttyp gehören die Feature Types `City, Island und Mountain`. Flächen und Linien werden dagegen durch die Linientypen verkörpert. Sie bestehen aus mehreren Linien die beim Rendern eine Fläche (`Country, Province, Lake`) oder eine Linie (`River`) repräsentieren. Linienzüge speichert das `GeoObject` in `Polyline` und übernimmt das Rendern eigenständig um unabhängig von dem verwendeten `Layer` arbeiten zu können. Wegen der Ungewissheit über den Zustand von Linienzügen können Objekte aus mehreren Linienzügen bestehen. So werden die Linien separat gerendert, was für die Darstellung nur einen geringen Unterschied bedeutet. Es könnten Lücken entstehen, die in aller Regel sehr klein ausfallen, wenn Start- und Endpunkte von zwei benachbarten Linen nicht identisch sind.

`GeographicText` repräsentieren die textuelle Auszeichnung des Namens. Das `GeoObject` unterscheidet bei der Platzierung zwischen der Art des eigenen Objektes. Wenn es sich um einen Punkttyp (z.B. City) handelt, platziert es den Text genau auf die hinterlegten Koordinaten. Bei Linientypen (z.B. Province) positioniert es die Auszeichnung dagegen in die Mitte des `Sector`. Für die Validierung bzw. Vervollständigung der Koordinaten, die primär aus Mondial bezogen werden, nimmt das Objekt zusätzlich die Daten aus dem OSM-Import auf. Wegen unterschiedlicher Schreibweisen kann es dazu führen, dass der Import mehr als eine Koordinate bereitstellt. Um einen möglichen Datenverlust zu vermeiden, speichert das Objekt alle Koordinaten ab und zeigt diese bei Bedarf an.

Wie bereits erwähnt, hält das `GeoObject` die Möglichkeit bereit die Daten aus Mondial und dem OSM-Import zu speichern. Um diese drei Arten der Datenherkunft besser abzugrenzen, sieht das Objekt drei verschiedene Farben vor. Die Erste markiert Objekte deren Daten aus Mondial stammen. Die Zweite zeichnet alles aus was in Mondial keine Daten für die Darstellung besitzt und mit OSM-Informationen vervollständigt wird. Die Dritte markiert Objekte, die zusätzlich aus OSM stammen und mit Mondial validiert werden können.

Die natürliche Ordnung der Objekte in den Listen geschieht nach dem Namen, weil dieser in den meisten Fällen den Primärschlüssel repräsentiert. Es gibt Ausnahmen wie z.B. `Country`, die nach anderen Kriterien sortiert werden müssen. Dazu muss die Methode

```
public int compareTo(GeoObject o) { ... }
```

überschrieben werden.

Feature Types

In Mondial befinden sich die Feature Types:

Continent, Country, Province, City, Sea, River, Lake, Desert, Island, Mountain

Andere Entitäten, die nur der Abbildung verschiedener Beziehungen zwischen den oben genannten Feature Types dienen, sind als Referenzen in die Objekte integriert. Im nächsten Abschnitt folgen die Feature Types mit ihrem referenziellen Abhängigkeiten.

Continent Country(0..*)

Country Continent(0..*), Province(0..*)

Province City(0..*), Sea(0..*), River(0..*), Lake(0..*), Desert(0..*), Island(0..*), Mountain(0..*)

City Province(0..1), River(0..*), Sea(0..*), Lake(0..*), Island(0..1)

Sea Province(0..*), City(0..*), Sea(0..*), River(0..*), Lake(0..*), Island(0..*)

River Province(0..*), City(0..*), Sea(0..*), River(0..*), Lake(0..*)

Lake Province(0..*), City(0..*), Sea(0..*), River(0..*), Lake(0..*), Island(0..*)

Desert Province(0..*)

Island Province(0..*), City(0..*), Sea(0..*), Lake(0..*), Mountain(0..*)

Mountain Province, Island(0..1)

Die Referenzen sehen vor, dass jedes Objekt nur mit seinem direkten Vorgänger bzw. Nachfolger verbunden ist. Beispielsweise steht eine `City` eigentlich mit einer `Country` in Beziehung. Diese Relation ist aber indirekt über die `Province` schon vorhanden.

Das Rendern der `GeographicText` und der `Polyline` ist so implementiert, dass das Objekt in erster Linie die Daten aus Mondial darstellt. Wenn aus Mondial keine Daten für die Visualisierung vorhanden sind, werden sie mit Daten aus OSM vervollständigt. So wird gewährleistet, dass für die Visualisierung der Großteil der Daten zur Verfügung steht.

Für den Fall, dass die Daten in Mondial vorhanden sind, extrahiert die Anwendung zusätzlich Daten aus OSM. Sie werden bei aktiver Validierung zusätzlich gerendert um Abweichungen mit dem Datenbestand von Mondial aufzeigen zu können.

Rekursion beim Rendern

Um Objekte darzustellen, die in Relation zueinander stehen, existiert die Rekursionstiefe. Je nach angegebener Tiefe ist es möglich die referenzierten Objekte in verschiedenen Stufen zu rendern. Um diese Eigenschaft zu nutzen muss der implementierte Layer nur die Methoden

```
public void render(DrawContext dc,
        boolean validate, int rek);
```

und/oder

```
public void setTexts(HashSet<GeographicText> list,
        boolean validate, int rek);
```

mit der gewünschten Rekursionstiefe aufrufen. Das Objekt rendert automatisch alle referenzierten Objekte bis zur angegebenen Tiefe.

Wegen zu großer flächenmäßiger Ausbreitung der in Beziehung stehenden Objekte, rendern `Country` und `Sea` nur Objekte, die semantisch unter ihnen angeordnet sind. `Country`-Objekte rendern keinen `Continent`, weil dieser bei bestimmten Ländern aus zwei Referenzen bestehen kann. Das könnte beispielsweise dazu führen, dass "Europa" und "Asien" angezeigt werden, was aber nicht Ziel der Anfrage ist. Eine `Sea` vermeidet es ihre referenzierten Nachbarmeere darzustellen, weil sonst sehr schnell alle Meere dargestellt sein könnten.

Das folgende Beispiel präsentiert die Auswahl der `Country` "Austria" mit der Rekursionstiefe "2". Es ist zu erkennen, dass zusätzlich zu `Country` ebenfalls die `Provinces`, `Cities`, `Rivers`, `Lakes` und `Mountains` angezeigt werden.

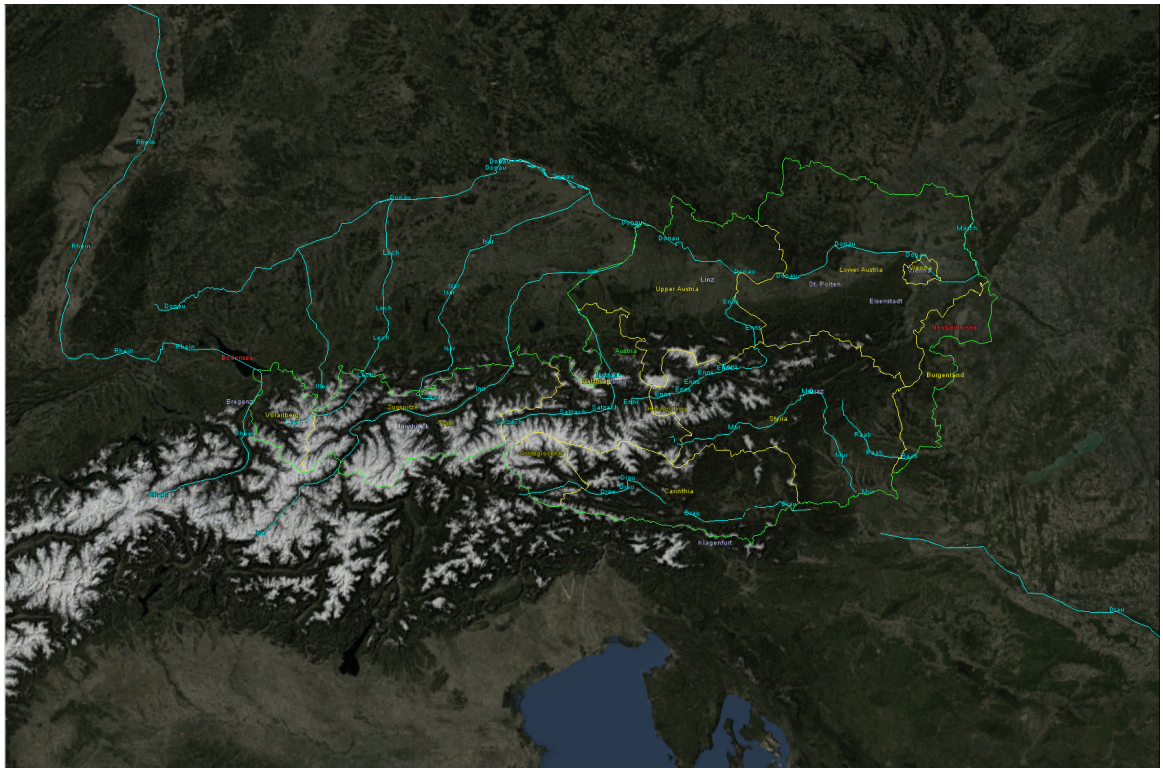

Abbildung 4.2: Österreich mit Rekursionstiefe 2

4.2.3 MondialLayer

Der MondialLayer bildet die Kernkomponente dieser Applikation. Der Layer stellt die Funktionalität bereit, verschiedene Features auszuwählen und diese auf dem 3D-Globus anzeigen zu lassen. Die Auswahl kann über das zugehörige grafische Interface (GUI) getätigt werden, das mit der Aktivierung des Layers sichtbar wird. Für jede Auswahl führt der Layer nebenläufig einen zusätzlichen Import aus OSM durch. Je nach Art der Auswahl werden dabei auch mehrere Objekte gleichzeitig importiert.

Die folgende Abbildung soll einen kleinen Überblick über die Zusammenhänge geben. Aus Gründen der besseren Übersichtlichkeit wird immer nur der letzte Teil des Paketnamens angegeben. Es sei aber darauf hingewiesen, dass sich alle Pakete in dem gemeinsamen Paket `de.uni_goettingen.informatik.worldwind` befinden.

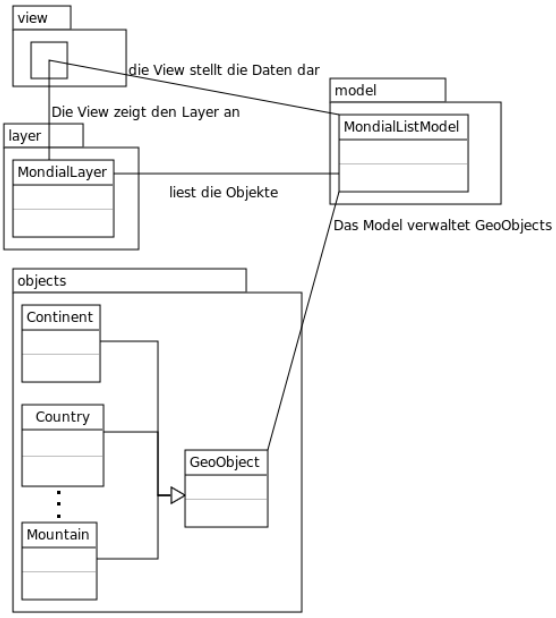

Abbildung 4.3: Übersicht World Wind

Für die Darstellung aller grafischen Komponenten wird das Paket `view` benötigt. Hier wird sowohl die Grundanzeige für die Elemente aus World Wind verwaltet als auch die Anzeige dieses Layers und seiner zugehörigen GUI. Für die benötigten Feature-Listen wird eine Datenverwaltung angelegt, die im `MondialListModel` implementiert ist. Die Verwaltung basiert auf dem `GeoObject` um unabhängig von dem verwendeten Feature Type arbeiten zu können.

Der `MondialLayer` arbeitet in diesem Modell als Kommunikationschnittstelle. Jede Änderung des `WorldWindowGLCanvas` veranlasst den Layer allen gewählten `GeoObjects` in der `MondialListModel` mitzuteilen, dass sie sich rendern sollen.

Die Kommunikation zwischen der GUI und dem MondialLayer wurde mit Hilfe von JavaBeans realisiert. Diese Technik bietet die Möglichkeit Interaktionen zwischen den Komponenten auf Basis des `Event-Listener-Prinzips` zu realisieren. Die Interaktion mit den World Wind-Komponenten wird über die vordefinierte Schnittstelle eines Layers realisiert.

Die reine Auszeichnung der `GeographicText` ist für die Anzeige hinreichend, aber es muss eine Möglichkeit zur Darstellung von Zusatzinformation für die Objekte gefunden werden. World Wind bietet für solche Textauszeichnungen die Möglichkeit eine `GlobeAnnotation` im Canvas zu platzieren. Um die Annotations nur bei Bedarf anzuzeigen, implementiert der `MondialLayer` den `PositionListener`. So kann er mit jeder Mausbewegung überprüfen, ob sich einer der anzuzeigenden `GeographicText` in unmittelbarer Nähe des Mauszeigers befindet. In Abhängigkeit von der Entfernung des Blickpunktes

zum Globus passt der Layer die Distanz zum genauen Punkt des `GeographicText` an.

Ein alternativer Ansatz für die Erzeugung der Annotations ist es, eine eigene Erweiterung der `GeographicText` zu implementieren, die den Text "pickable" machen. Über diese Eigenschaft könnten anschließend `MouseListener` auf die `Events` reagieren, die beispielsweise durch einen Mausklick ausgelöst werden. Der Mehraufwand für die Implementierung der benötigten Funktionalität steht aber in keinem akzeptablen Verhältnis zu seinem Nutzen.

Eine andere Möglichkeit ist es, die Annotations versteckt auf dem Canvas zu platzieren und sie erst bei einem bestimmten Ereignis anzuzeigen. Diese Variante liefert zwar das gewünschte Ergebnis, aber es kommt mit steigender Anzahl von Annotations dazu, dass die Darstellung ins Stocken gerät.

GUI

Die zum `MondialLayer` gehörende GUI umfasst die Anzeige- und Auswahlmöglichkeiten für die verschiedenen Features. Für jeden Feature Type existiert eine separate Liste um eine gezieltere Suche nach Objekten zu realisieren. Anderenfalls könnte es dazu kommen, dass es zwei Features mit gleichem Namen gibt.

Die Umsetzung einer Auswahl ist im `DnDListener` [31] implementiert. Wenn eine Auswahl getätigt wird, sorgt der `DnDListener` dafür, dass die angefragten Objekte der Warteschlange für den OSM-Import hinzugefügt werden. Über das Suchfeld können zusätzlich Objekte gesucht werden, die nicht in Mondial enthalten sind. Diese Funktionalität erweitert die anzuzeigende Datenmenge auf die in OSM beinhalteten Features.

Weitere Optionen auf der GUI sind die Angabe der Rekursionstiefe und die Aktivierung oder Deaktivierung der Validierung von Mondial mit OSM. Aus Laufzeitgründen wurde die maximale Rekursionstiefe auf Stufe drei limitiert.

4.3 Anmerkungen

4.3.1 Probleme

Aufgrund der 3D-Beschleunigung von World Wind kann es beim Rendern großer Datenmengen zu Problemen bei der Darstellung kommen. Je nach Hardware können diese Probleme in ihrer Intensität variieren. Deshalb wurde versucht bei der Umsetzung möglichst ressourcensparend vorzugehen um die Anwendung einem möglichst großen Anwenderkreis zugänglich zu machen.

Um die Anzahl der darzustellenden Objekte nur auf den sichtbaren Bereich zu reduzieren wird für jedes Objekt, das aus Linien besteht, eine Boundingbox angelegt. Die da-

[31] de.uni_goettingen.informatik.dbis.WorldWind.Controller.DnDListener

mit mögliche Sichtbarkeitsprüfung reduziert die Anzahl der anzuzeigenden Objekte mit steigender Zoomstufe. Auf niedrigen Stufen besteht der sichtbare `Sector` jedoch aus sehr großen Kartenausschnitten, wodurch diese Maßnahme ihre Vorteile nicht voll ausreizen kann.

Ein weiteres Mittel zur Reduzierung des Rechenaufwandes ist es, Linienzüge nur zu rendern, wenn der Globus sich nicht bewegt. So stellt die Anwendung trotz großer Datenmengen eine flüssige Bewegung dar. Um trotzdem eine Orientierung zu ermöglichen sind die Textauszeichnungen immer sichtbar.

Die Auszeichnung von Linienzügen und ihren eingeschlossenen Flächen stellt weiterhin ein Problem da. Es wird versucht die Auszeichnungen immer in die Mitte der Boundingbox des Objektes zu platzieren. Objekte, die durch die Suche falsche bzw. zu viele Ergebnisse enthalten, spannen einen zu großen `Sector` auf, was eine Fehlplatzierung der Auszeichnung zur Folge hat. Dieses wird dadurch hervorgerufen, dass der OSM-Import bei der Suche unterschiedliche Schreibweisen und das Enthaltensein überprüft. So kann beispielsweise bei der Suche nach der "Havel" auch der "Oder-Havel-Kanal" gefunden werden. Aber auch Objekte, die aus nicht zusammenhängenden Teilen bestehen, können dieses Verhalten hervorrufen. Beispielsweise besitzen die Niederlande Inseln in der Karibischen See. Wegen der großen Entfernung wird die zugehörige Boundingbox sehr breit, was zur Folge hat, dass die Auszeichnung für die Niederlande mitten auf dem Atlantischen Ozean platziert wird.

4.3.2 Ausblick

Für kommende Erweiterungen könnten die Laufzeitprobleme weiter minimiert werden, indem für Darstellung anstelle der `Polyline` Grafiken verwendet werden, die nur auf der Oberfläche abgebildet werden müssen. Der Layer `Political Boundaries` zeigt erste Ansätze für die Projizierung der Grafiken auf den 3D-Globus. Die Implementierung für die Projizierung ist wahrscheinlich mit geringen Aufwand zu realisieren. Die Erzeugung der Grafiken dagegen würde einen erheblichen Mehraufwand bedeuten. Implementierungen dieser Lösung könnten auf drei verschiedenen Techniken basieren.

Eine Möglichkeit ist es die Grafiken dynamisch zu erzeugen. Der Vorteil dabei wäre, dass mehrere Features in einer Grafik enthalten sein könnten. Die Erzeugung neuer Grafiken führt eventuell wieder Laufzeitprobleme. Es ist denkbar, dass die Anwendung ins Stocken kommt, weil die Wartezeit für die Berechnung der Grafiken zu groß wird. Jedes Mal, wenn Objekte entfernt oder hinzugefügt werden, wird die Grafik neu erstellt.

Der nächste Ansatz würde vorsehen, dass für jedes Feature eine eigene Grafik vorgehalten wird. Der große Nachteil dabei wäre der immense Speicherverbrauch für die verschiedenen Grafiken. Dafür spricht aber, dass die Grafiken sofort zur Verfügung stehen und kein Aufwand betrieben wird um die Grafiken zu erzeugen. Je nach Herkunft (lokal oder aus dem Internet) ist der Geschwindigkeitsvorteil für die Zulieferung der Grafiken

hervorzuheben. Gegen dieses Verfahren würde allerdings sprechen, dass mehr Grafiken verarbeitet werden müssten. Daraus könnten wiederum Laufzeiteinbußen bei der Darstellung resultieren. Es müsste also erst einmal überprüft werden, ob dieses Verfahren der aktuellen Implementierung überlegen ist.

Der dritte Ansatz ist eine Kombination aus den zwei vorhergehenden. Wie im zweiten werden die Grafiken für jedes Feature vorgehalten. Anschließend würden die Grafiken je nach Anfrage miteinander kombiniert werden, um wieder nur eine bzw. nur wenige Grafiken für das Rendern bereitzustellen. Je nach verwendetem Format könnten so die Vorteile beider Techniken sehr gut miteinander kombiniert und die Nachteile weitestgehend minimiert werden.

Das zweite und damit auch das dritte Verfahren könnten gleichzeitig das Problem mit der fehlerhaften Auszeichnung von Features beheben. Denkbar wäre die direkte Integration der Auszeichnungen in die Grafiken. Durch die unabhängige Erstellung der Grundgrafiken könnten diese optimal platziert werden, ohne dass die erwähnten Fehlinterpretationen auftreten.

Genauere Aussagen über das Verhalten können aber erst mit Hilfe von Prototypen getätigt werden. Vermutlich würde das dritte Verfahren zum gewünschten Laufzeitgewinn führen und so in einer noch besseren Benutzbarkeit der Anwendung resultieren.

5 Benutzerhandbuch

5.1 Apache Ant

Dem Projekt sind zwei Buildfiles `build.xml` zur Ausführung mit Apache Ant [1] beigefügt. Mit Hilfe dieser können die Binärdateien, die Jar-Dateien und die Javadoc-Dateien erstellt werden. Es stehen folgende Targets zur Auswahl bereit:

ant clean bisher erstellte Dateien werden gelöscht.

ant init erstellt die Verzeichnisse für die Binärdateien, das Jar-File und die Javadoc-Dateien.

ant compile erstellt die Binärdateien.

ant jar erstellt das Jar-File.

ant doc erstellt das Javadoc für das Projekt.

ant build erstellt die Binärdateien, das Jar-File und die Javadoc Dateien.

Das Default Target ist `ant build` und wird automatisch beim Aufruf von `ant` ausgeführt.

5.2 Erzeugung der Feature Type Dateien

Zum Erzeugen der Feature Type Dateien dient das Programm `osmFeatures`. Es steht im gleichnamigen Verzeichnis zur Verfügung. Durch die Ausführung des beiliegenden Buildfiles `build.xml` mit dem Aufruf

```
ant
```

wird das Unterverzeichnis `jar` samt der Jar-Datei erzeugt.

Zusätzlich wird ein Datenauszug des OSM-Projektes benötigt. Die Datei `planet.osm.bz2` kann unter http://planet.openstreetmap.org heruntergeladen werden.

Über den Aufruf

```
java -jar osmFeature.jar <planet.osm.bz2>
```

[1] http://ant.apache.org/

unter Angabe der heruntergeladenen OSM-Datei, werden die Feature Type Dateien `type1`, ..., `type7` im aktuellen Arbeitsverzeichnis erzeugt.

5.3 World Wind

5.3.1 Systemanforderungen

Für die Verwendung von World Wind werden folgende Anforderungen an das System gestellt:

- eine OpenGL-fähige Grafikkarte
- Sun Java Runtime Enviroment ab Version 5 (1.5)
- ca. 150 bis 200 MB Speicherplatz für die Installationsdateien
- zusätzlich benötigt der Cache von World Wind im Benutzerverzeichnis ca. 500 bis 1000 MB Speicherplatz

5.3.2 System Einstellungen

World Wind verwendet für die Darstellung S3TC-komprimierte Texturen. Auf Microsoft Windows-basierten und Mac OSX-Betriebssystemen sollte dieses Format keine Schwierigkeiten machen. Auf Unix/Linux-Betriebssystemen könnte die benötigte Softwareunterstüzung fehlen. Aus diesem Grund wird DRIconf benötigt um die Texturen trotzdem zu verwenden.

Die Anwendung DRIconf ist eine Verwaltungsapplikation für die Direct Rendering Infrastruktur. Die Performance und die optische Qualität der OpenGL-Treiber passt sie individuell an. Die Einstellungen können systemweit, benutzerabhängig oder anwendungsabhängig angelegt werden.

Unter Ubuntu befindet sich die Anwendung in den "universe" Repositories. Für andere Linux-Systeme sind die Sourcen auf `http://dri.freedesktop.org/wiki/DriConf` verfügbar.

Je nach Startmodus legt DRIconf beim Systemstart eine neue Konfigurationsdatei an, in der es die Einstellungen speichert. Auf dem Reiter "Bidlqualität" befindet sich die zu aktivierende Option "Aktiviere S3TC Texturkomprimierung auch wenn die nötige Softwareunterstützung fehlt". Ohne die Aktivierung kann es auf einigen System dazu führen, dass World Wind die Texturen für den Globus nicht darstellen kann.

Der Datenimport benötigt das "mondial.xml" und die "type*"-Dateien für die `osmFeatures` Bibliothek und erwartet, dass sie sich in dem Verzeichnis `data` befinden. Das Installationspaket hält bereits erste Versionen bereit, die bei Bedarf aktualisiert werden können.

5.3.3 Ausführung

Die Anwendung erwartet, dass sie immer aus dem Verzeichnis ausgeführt wird, in dem sich das Jar-File befindet. Anderenfalls könnte es dazu kommen, dass die Anwendung die Pfade nicht richtig auflösen kann und keine oder falsche Daten lädt.

Der Befehl um die Anwendung zu starten lautet

```
java -jar mondial.jar
```

wenn die Bibliotheken sich im Unterverzeichnis der JVM (z.B. `../jvm/java-6-sun/jre/lib/ext/`) befinden. Anderenfalls muss der `java.library.path` jedes Mal beim Aufruf erweitert werden mit

```
java -Djava.library.path=$path:lib/ -jar mondial.jar
```

Nach dem Start der Anwendung erscheint folgende Oberfläche.

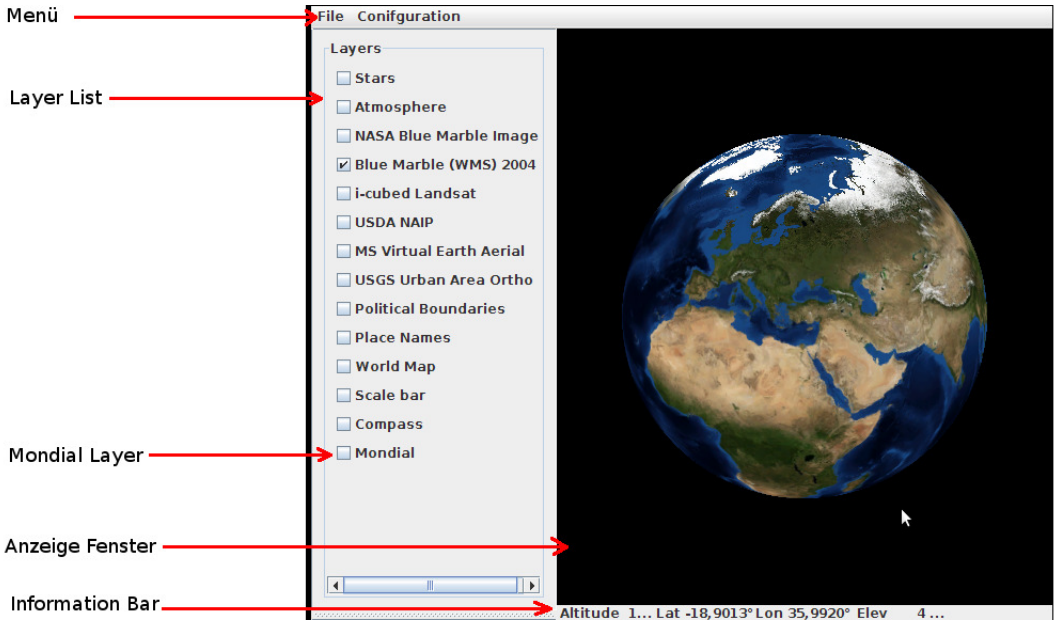

Abbildung 5.1: World Wind Graphical User Interface

Auf dem Fenster befindet sich ein Menü für die die individuelle Anpassung der darzustellenden Objekte. Das Fenster unterteilt sich in die zwei Bereiche "Anzeige" und "Layer" Liste. In der Anzeige wird der 3D-Globus dargestellt. Über die Layer Liste können die verschiedenen Layer aktiviert oder deaktiviert werden.

Mit der Aktivierung des MondialLayers wird die zugehörige Oberfläche eingeblendet. Der grundlegende Aufbau ist so gestaltet, dass sich im oberen Bereich die Listen mit den

verschiedenen Feature Types befinden. Im Zusammenspiel mit dem darüber liegenden Suchfeld wird in der aktiven Liste nach Objekten gesucht. Die Übersicht aller ausgewählten Objekte befindet sich in einer weiteren Liste, die sich unterhalb der Feature Type Listen ansiedelt. Weitere Optionen sind die Validierung mit Daten aus OSM und die Einstellung der Rekursionstiefe für das Darstellen von Objekten und ihrer relationalen Features. So können z.B. Länder und ihre zugehörigen Provinzen dargestellt werden, obwohl nur das Land selbst ausgewählt ist.

Auf der folgenden Abbildung werden noch einmal die einzelnen Komponenten des Mondial Layer für das bessere Verständnis markiert.

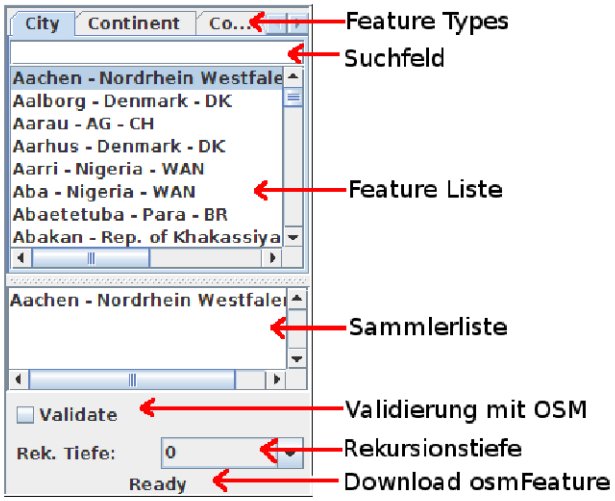

Abbildung 5.2: Graphical User Interface des Mondial Layer

Informationen über den Verarbeitungszustand des OSM Imports kann dem untersten Panel entnommen werden.

Tastenkombination

Für die bessere Benutzbarkeit existieren folgende Tastenkombinationen:

ALT+(0..9) direkte Auswahl eines Feature Types,

STRG+f setzt den Fokus auf das Suchfeld des gerade aktivierten Feature Types,

STRG+a wenn der Fokus auf einer Liste liegt, werden alle Objekte markiert,

STRG+q beendet die Anwendung.

Allgemeine Steuerung World Wind

schwenken Linke Maustaste gedrückt halten und anschließend die Maus in die gewünschte Richtung bewegen.

zoomen Mit Hilfe des Scrollrades der Maus.

Blickwinkel ändern Rechte Maustaste gedrückt halten und die Maus hoch oder runter bewegen oder die "Page-Up" und "Page-Down" Tasten verwenden.

rotieren Die rechte Maustaste gedrückt halten und anschließend die Maus nach links oder rechts bewegen

Reset Die Rotation wird mit der Taste "n" wieder hergestellt. Mit der Taste "r" wird zusätzlich der Blickwinkel wieder zurückgestellt.

Auswahlmöglichkeiten

Um ein Feature auszuwählen existieren verschiedene Möglichkeiten. Die erste Möglichkeit basiert auf der Steuerung mit der Maus. Die Auswahl kann ein oder mehrere Features umfassen, die anschließend per Drag&Drop auf die Sammlerliste gezogen werden. Eine andere Vorgehensweise sieht vor, dass Objekte über die Textsuche selektiert werden. Bei erfolgreicher Suche bewirkt die "Enter" Taste die Auswahl des gesuchten Objektes und übernimmt es in die Sammlerliste. Sollte die Suche kein Feature in Mondial finden, wird versucht den eingegebenen Namen in OSM zu finden und anzuzeigen. So erweitert sich der Suchraum von Mondial auf OSM.

Farben

Über das Menü "Configuration $\rightarrow$ change color" können für jeden Feature Type eigene Farben festlegt werden. Die Feature Types teilen sich in drei verschiedene Arten auf, die jeweils eine eigene Farbe besitzen. Die erste Art repräsentiert alle Daten, die direkt aus Mondial stammen. Die zweite sind die Daten aus dem OSM Import. Die dritte vertreten die Daten für die Validierung, das bedeutet die Informationen können aus Mondial und OSM bezogen werden.

6 Lizenzen

6.1 Apache Commons Compress

Die Bibliothek Apache Commons Compress steht unter der Apache License Version 2.0[2]. Folglich kann dieses Projekt die Software frei verwenden. Ihm muss aber eine Kopie der Lizenz beigefügt sein.

6.2 OpenStreetMap

OpenStreetMap besteht aus freien Daten, die gemäß der Lizenz Creative Commons Attribution-ShareAlike 2.0 (CC-BY-SA) [5] verfügbar sind.

6.3 World Wind

Das Java SDK von World Wind steht unter dem NASA Open Source Agreement v 1.3 [13]. Folglich kann dieses Projekt die Software frei verwenden, unter folgenden Voraussetzungen:

- Es muss eine Kopie der Lizenz beiliegen
- Die Anwendung muss frei zur Verfügung gestellt werden
- Der Quellcode muss frei verfügbar sein
- Eigene Änderungen müssen selbst entwickelt werden und dürfen von keinem Drittanbieter stammen

Die Free Software Foundation[1] erkennt diese Lizenz aufgrund des letzten Punktes als keine freie Lizenz an, weil die Open Source Entwicklung davon lebt eigenen Quellcode mit anderen zu kombinieren.

[1] http://www.gnu.org/licenses/license-list.html

7 Fazit

Für die Visualisierung geografischer Daten bietet sich eine Reihe von Systemen an. Es zeigt sich, dass der Funktionsumfang in allen Fällen das Zeichnen grafischer Primitive ermöglicht. So ist die Darstellung beliebiger geografischer Objekte möglich. Die Wahl des verwendeten Systems lässt sich daher ausschließlich auf Grundlage der Rahmenbedingungen treffen.

Die Wahl der Datenquelle hat hingegen einen deutlich größeren Einfluss auf die Güte der Visualisierung und Validierung. Für die optimale Repräsentation des Mondial Datenbestandes wäre eine Quelle nötig, die alle geforderten Feature Types in ausreichendem Umfang enthält. Viele der aktuellen freien Projekte in diesem Bereich erreichen den geforderten Umfang nicht. Dies ist zum einen auf den langwierigen Prozess der Datenerfassung, zum anderen auf die Zielsetzung der Projekte zurückzuführen.

Quellenverzeichnis

[1] Apache Commons Compress, http://commons.apache.org/compress.

[2] Apache License, http://www.apache.org/licenses/LICENSE-2.0.html.

[3] Christian Ullenboom, *Java ist auch eine Insel*, 8. Auflage.

[4] COM API von Google Earth, http://earth.google.com/comapi/.

[5] Commons AttributionShareAlike 2.0, http://creativecommons.org/licenses/by-sa/2.0/.

[6] Component Object Model, http://www.microsoft.com/com/default.mspx.

[7] GeoNames Projekt, http://www.geonames.org.

[8] Google Earth, http://earth.google.com/intl/de/.

[9] Java Binding für die OpenGL Spezifikation, http://jogamp.org/jogl/www/.

[10] Lesson: Using Swing Components, http://download.oracle.com/javase/tutorial/uiswing/components/index.html.

[11] Mapnik, http://mapnik.org.

[12] Marble Projekt, http://edu.kde.org/applications/all/marble.

[13] Nasa Open Source Aggrement, http://worldwind.arc.nasa.gov/worldwind-nosa-1.3.html.

[14] Open Geospatial Consortium, http://www.opengeospatial.org.

[15] OpenGIS Web Feature Service (WFS) Implementation Specification, http://www.opengeospatial.org/standards/wfs.

[16] OpenLayers Projekt, http://openlayers.org.

[17] OpenStreetMap, http://www.openstreetmap.org.

[18] OpenStreetMap Wiki, http://wiki.openstreetmap.org.

[19] The Industry's Foundation for High Performance Graphics, http://www.opengl.org/.

[20] White Paper: S3TC compression technology, http://www.computerweekly.com/Articles/1999/10/25/178983/white-paper-s3tc-compression-technology.htm.

[21] World Wind Projekt, http://worldwind.arc.nasa.gov/java.